Usage Analysis & Demonstrators - Version 2.0

Alain Boulze, Stéphane Bagnier, Julien Forrest, Sébastien Jourdain, Mohammed El Jai, Claude Meynier, Jérôme Besnainou, Marc Dutoo, Marcel Arrufat Arias, Adrian Mos, et al.

SCOrWare Project

WP3 Specifications

Usage Analysis & Demonstrators

Version 2.0

Funded by

AGENCE NATIONALE DE LA RECHERCHE
ANR

http://www.agence-nationale-recherche.fr

Date	March, 10, 2009
Deliverable type	Specification
Version	2.0
Status	Revision
Work Package 3	Usage Demonstrators
Access Permissions	Public

Editor

INRIA Tuvalu	Alain Boulze

Authors

Artenum	Stéphane Bagnier
	Julien Forest
	Sébastien Jourdain
EBM Websourcing	Mohammed El Jai
	Claude Meynier
Edifixio	Jérôme Besnainou
Open Wide	Marc Dutoo
Thales Communications SEA/TAI	
	Marcel Arrufat Arias
INRIA Tuvalu	Adrian Mos
	Guillaume Vaudaux-Ruth

Contributors

SCOrWare Consortium

Table of Contents

Index of illustrations

Index of tables

1 INTRODUCTION

This second version of the «Usage Analysis and Demonstrators » document mainly presents four case studies done during the second part of the SCOrWare project:

- (Task 3.1) Component and service-oriented architecture in the Scientific Software field (improvements of works done during the first year)
- (Task 3.2) SCA as a SOA design methodology in the domain of CDE (Collaborative Development Environment). Following the withdraw of one of the partners (eXo Platform, provider of an open-source portal solution) during the first year, some changes have been decided during the second part of the project and an alternative demonstrator has been designed.
- (Task 3.3) How SCA contributes to reusing and enriching software components. Following the first year project's review, this scenario has been reinforced, and is the major demonstrator for the SCOrWare platform in the field of enterprise business applications.
- (Task 3.5) Using the SCOrWare platform and a component-oriented architecture in the context of a network monitoring system. A new partner (Thales Communications, in collaboration with Open Wide and EBM Websourcing) has joined the SCOrWare consortium during the second part of the project, following the withdraw of Amadeus.
- *Note:* the works done during the first year of the project around a service orchestration platform in the domain of travel reservation systems (Task 3.4) have been stopped, because of the withdraw of the main actor of this usage scenario (i.e. Amadeus).

In order to achieve the methodological work to be done in the context of the Task 3.3 ("reusing and enriching software components"), the development of the usage demonstrators has been completed by collaborative sessions between some partners involved in the Work Package 3. This collaborative work, led by EBM Websourcing, proposed during the second part of the project an in-depth analysis of how the demonstrators have been designed and developed, a better understanding of what developers have done, and how the SCOrWare platform is used.

This specification presents for every case study:

- motivation, business context, system context
- functional specification, usage scenarios
- architecture, technical design and specification
- coverage of the demonstrator (vs. SCA specification and SCOrWare technical platform)
- lessons learned, methodology, best practices, demonstrators' results

2 METHODOLOGICAL THOUGHTS

The works done around methodology in the context of the Task 3.3 have been conducted to help to identify how SCA and the SCOrWare platform could be used within a global SOA approach, as perceived and interpreted from a market point of view, and in particular from a first analysis done from the commercial competitors and offerings.

In a first stage of this work, we tried to define what a methodology is, and we got the following proposal:

Considering a problem "p" to be solved and a solution "s", we use a "SOA/SCA methodology" to build a solution "s", and we define "m" as the set of the tasks and their description conducting from "p" to "s" ("p->s"). We define "methodology" as an extrapolation of "m",i.e.:
p € P (P is a set of problems with a "p" type), s € S (S is a set of solutions using SOA/SCA for problems with a "p" type), m € M (M is a set of best practices for proposing a solution S solving a

problem P using SOA/SCA).

Such an ambitious and global objective has arisen several difficulties and issues, according to the following considerations:

- differences in considering what SOA is, according to the position, role and motivation of the partner involved in the SCOrWare project: academic or industrial, middleware or application developer, software editor or integrator, technologist or application, business aware, developer, designer, architect or business analyst
- level of abstraction and models handled by the different actors in the SCOrWare project: component level (such as in Fractal), infrastructure and middleware level (such as protocols, bindings, assembling), Java programing level (policies, properties), application architecture level (such as using SCA as an assembly of business components), process or workflow level (such as in environments like Eclipse JWT, Java Workflow Tooling or OW2 Scarbo, SOA ready SCA powered Eclipse BPM tooling), application or business level (considering the abstraction of the business application itself), ontology and semantics
- lack of real business features in the usage demonstrators as developed with the SCOrWare platform, as most of the usage demonstrators are technology-oriented
- difficulties for the developers of the usage demonstrators to be observed and their work to be in-depth analyzed and criticized (to be understood under the positive meaning of this wording), lack of resources and time too.

Finally, this methodological work has shown that such a methodology hasn't been really applied and used for the usage demonstrators, as the formalization of the business problem was very weakly treated, and as the usage demonstrators' domains are very close from technologist concerns. Moreover, even if there were intentions and actions for defining the services (following SOA concepts) of a New Generation Forge (in the context of the task 3.2), this approach hasn't been finally concretized and demonstrated by the demonstrator itself.

Nevertheless, the methodological work which has been done has proven the great interest of SCA to facilitate a common understanding and sharing of the technological architecture for developing the solutions. In particular, the SCA graphical representation and the related concepts allow to "draw" and visualize the composition of the application architecture ("SCA Assembly") to be implemented and deployed. Illustration 12 (for task 3.1), Illustration 23 & Illustration 25 (for task 3.2), Illustration 44 (for task 3.3), and Illustration 47 (for task 3.5) illustrate this main feature of SCA in terms of methodology. Besides, the demonstrator as developed in the task 3.3 has shown the benefit of having a "SCA container" integrated into a JBI ESB platform such as PEtALS, for composition of services, and integration of SCA and JBI.

We propose the following chart as a result of the observation and work done in this methodological work across the different usage demonstrators developed in the task 3.

Usage Demonstrators	***Business Context***	***Application Architecture***	***Technical Architecture***	***Programing Model (Java)***	***Deployment & Volumetric***
T3.1 [Scientific Computing]	No formalization	UML & SCA Assembly	SCA Domain SCA Binding RMI & Web Service	Serialization for large data SCA Property ?	FraSCAti ***Vol.: large data***
T3.2 [Collaborative Development]	UML ***SOA Design tentative*** & relationship to BPMN	UML & SCA Assembly	SCA Domain SCA Binding Web Service	No SCA Policy SCA Property	Tuscany 1.3.2 & FraSCAti 0.5 Vol.: N/S
T3.3 [Corporate F/E	Ontology	UI mock-ups	SCA Binding Web Service	SCA Policy ?	Tuscany ***FraSCAti+***

Order Mgt linked with Back Office] ***[Enterprise Case Study & SCOrWare "Pilot"]***	(Semantic Trader & Composer)	UML & SCA Assembly ***Assembly Evolution***	EJB RMI JDO2 (JDBC) JBI (via PEtALS)	SCA Property ?	***PEtALS (JBI/SCA)*** ***Vol.: number of components, compilation & development time***
T3.4 [Travel Transaction Orchestration]	N/A	N/A	N/A	N/A	N/A
T3.5 [Network Monitoring]	No formalization	SCA Assembly ***Assembly Evolution Multi-compositions***	SCA Binding Web Service & JMS (via PEtALS)	SCA Policy ? SCA Property ?	***FraSCAti+ PEtALS (JBI/SCA)*** Vol.: N/S

Table 1: Methodological chart

This table describes the following methodological activities, and what are their relationships with the SCOrWare platform and works:

- *Business Context*: how the business domain has been described ? what formalization has been chosen ? impact of SOA at this stage ?
- *Application Architecture*: which description and formalization of the application architecture ? impact of SCA at this stage ? usage of SCA tool set ?
- *Programing Model, Technical Architecture*: which features of SCA and the SCOrWare platform have been used ?
- *Deployment and Volumetric*: which SCOrWare runtime platform has been used ? which volumetric and is it significant ?
- ***Main characteristics as demonstrated are highlighted (bold+italic) in this chart***

Besides this overall classification with relation to a unifying "methodological" approach, every usage demonstrator as described in this specification gives an overview of best practices, methodological aspects, and lessons learned from the design and the development of the demonstrators. Across the different usage demonstrators, this part of the specifications lets appear different points of view and practices as capitalized during the development of the demonstrators and their design.

3 A SCA APPROACH FOR SCIENTIFIC COMPUTING (Task 3.1)

Change history vs. version 1.0: the first version 1.0 (October, 2007) of the specification was mainly describing the business problem and a first theoretical analysis. This final version fully describes the demonstrator, its technical design and implementation.

3.1 Introduction, business / system context

3.1.1 Motivation

The 3.1 sub work-package addresses the open question of the use of component based and service oriented architecture in the scientific computing field.

Scientific computing is an advanced area of the computing industry in terms of computing capacity, amount of data processed and performance expectation. However, tailored software developments in

this field are not leaded by computing specialists then most of the useful codes are written by PhDs in science, researchers and engineers who stick to quite simple development methods and paradigms.

However, we see nowadays a progressive emergence of service-oriented architectures in the scientific field, especially in data processing. More and more research projects attempts to offer an access to their tailored tools through Web Services. In the domain of the exchange of complex data, Corba Bus have been explored at the late 90s. This technological evolution answers to a significant evolution of the market, especially for engineering application like in PLM processes or complex data handling (e.g CAD models, mesh, experimental data…). The geographical distribution of teams and experts, on one side, and the wish to control the diffusion of classified software or methods, on the other side, reinforce this need. One of the first examples could be the Salome platform [1] where each business components can be distributed remotely and the data exchange is performed on a CORBA bus. And more generally, the Integrated Modeling Environments (IMEs) able to remotely address the whole modeling process and offer tailored declinations (i.e adaptation of the GUI, integration of specific tools in composite application) introduce a new approach in mathematical modeling and numerical simulation.

In the present work, we make the choice to focus our study on the distribution of the visualization process of scientific data.

De facto most of 3D visualization tools present architectures that are close to service-based approach with the concept of visualization pipeline [2] [3]. In this concept, the whole processing is done by the composition of a set of basic filters (e.g. cutting plane). Processed scientific data are passed along this pipeline and 3D virtual reality objects, corresponding to the output data, are generated for the final rendering and visualization. Illustration 1 below illustrates this principle.

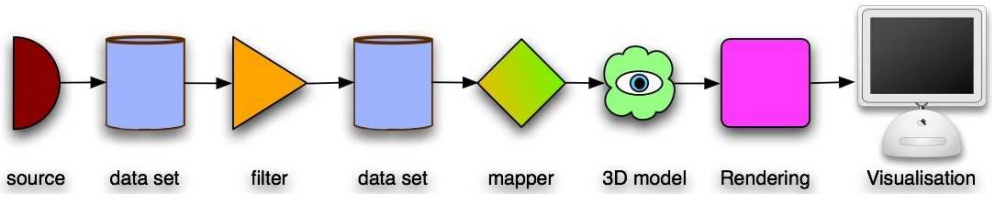


Illustration 1: canonical visualization pipeline

Data processing and 3D visualization outlines several simple problematic and critical issues that are actually characteristics of scientific applications more generally speaking.

Some of these issues are directly related to the processed data themselves:

- Data size: Processed data can become quickly very large and introduce sever constraints in terms of memory foot print and network bandwidth;
- Data complexity: Scientific data may present sever constraints in terms of complexity, due to their intrinsic structure (e.g. data fields deployed on mesh, temporal series…) or due to their implementation. Some data are, for instance, not easily serialisable due to the used technologies (e.g. Fortran or C/C++ native layer) or structure (cyclic references).
- Data formats and conversion: For both historical and technical reasons, there is no general consensus regarding a unique and standard data format for scientific data. This situation generally requires complex and CPU costly conversion processes.
- Performances: Generally speaking, due to the size of processed data and the complexity of the processing, scientific data processing induces severe performances constraints.

In the validation process of SCA for scientific computing, several experimentations were done. We focused our analyses on the following aspects:

- The ease of use for non software developers specialists,
- Existing service exposition,
- The flexibility in the usage and in composition of services,
- Cost in term of performance regarding the binding abstraction,

- The design issue.

3.1.2 RNTL SCOS/V3D context and additional contributions from SCA

This work was performed in parallel to the RNTL/SCOS project and, more specifically, its sub-project SCOS/V3D. SCOS/V3D aims to create a framework for the high throughput 3D scientific data visualization, distributed and collaborative.

SCOS/V3D follows the recent evolution of visualization tools where the visualization process is split into different steps. The data are processed on a remote and high performances server and the rendering is performed locally in a client application. One of the key innovations of SCOS/V3D is the possibility to access to the processing service through Web services, using light or semi-rich clients, and offer collaborative functionalities. This should allow displaying a large amount of data while offering a quite convenient access to the users. With respect to the existing solutions, several innovative choices have been done in SCOS/V3D:

- Let the data on the server and avoid all transfer of the raw data to the client side, to optimize network load and to simplify the requirements on the client workstation.
- Transfer of resulting data to the client under the form of a 3D virtual reality scene, using the X3D format to offer a better use of the client computing and memory capacities and to offer a 3D fluid navigation with less network communications.
- A high processing capacity by parallelizing the tailored processes on the server side.
- Offer the possibility to compose and test new processing artifacts locally and deploy them on the server side.

The architecture of SCOS/V3D is composed of several distinct functional services: a rich client application, a web server, a visualization server, a data server, and a processing server. Underneath, SCOS/V3D requires a high performance computing cluster to support the processing server, a Storage Area Network to support the data server and a couple of master nodes to support the resource management and scheduling. This architecture is summarized in the following schema:

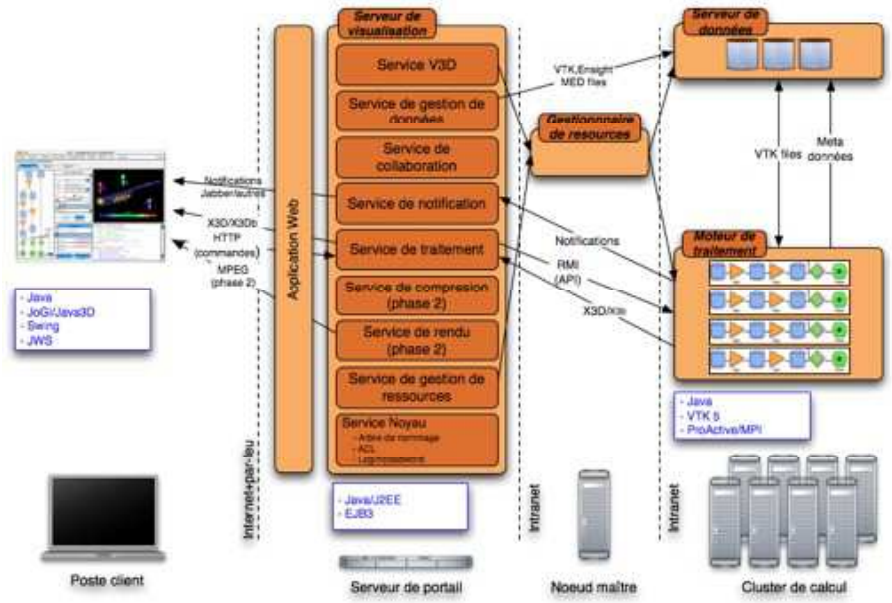


Illustration 2: global architecture of SCOS/V3D, based on a classic Web Services approach

The semi-rich client application has been developed in Java, using the JOGL library for the 3D rendering. An independent client library will be developed and will communicate with the visualization server through the HTTP or XMPP protocols, in order to pass through the industrial IS

constraints (e.g firewalls, proxy…).

The visualization server gather all the component needed to build-up a full 3D collaborative application, allowing the users to interact on the 3D scenes according to their access rights. Because all exchanges between the server and the client are done through request-answers based protocols (i.e. HTTP), a specific notification server has been developed [4], to allow the server to notify dynamically the client of the 3D scenes recomposition.

The processing server uses the VTK library [3] to process the data and to generate 3D scenes. These processes have to be efficiently balanced between the hardware nodes of the cluster in order to reach a high throughput capacity. The input datasets should be streamed directly from the data server and the output 3D scene should be streamed to the client through the visualization server. Both the settings and the composition of the processes should be defined from the client side, and communicated to the processing server thru an abstract set of instructions. These capacities make the processing server a versatile and highly re-configurable artifact

Last, one additional specificity of this demonstrator is that some tailored components are in a native language (C/C++) and their integration into a service oriented architecture presents an additional challenge, especially during the deployment phase.

The objective of the SCOS/V3D project was to mainly focus on the tailored layers and provide an operational solution in the duration of the project. For this reason and in spite the innovative functional choices, the chosen technical solution remains relatively classical.

It was chosen that the visualization server expose a lightweight web interface. This visualization server does not process any scientific data, and use both the data server and the processing server for this task. In practice, the SCOS/V3D implementation is a based on a simplified J2EE implementation. On the server side, the communications between the visualization server and the processing service is done using dedicated RMI/IIOP protocols.

In the frame of the ANR SCOrWare project, it is proposed to explore one-step further with the introduction of the complete Service Oriented Approach, using the SCA technology through a set of prototypes and demonstrators, the final objective being to provide a more versatile, scalable and adaptable visualization solution.

Reciprocally, the purpose of this set of experimentations is to check the global SCA specifications and how SCA can answer to the needs of the scientific community.

To address this objectives, a set of application scenarios have been identified, outlining potential bottlenecks or functional challenges, and a corresponding set of prototypes have been developed. Some of them are partially based on SCOS/V3D components for the tailored components, like the VTK based processing engine.

3.2 Theoretical analysis

In first step, a set of applicative scenarios has been identified, corresponding, on one side, to various practical situations possible in scientific data processing and, on the other side, in function of the coverage of SCA.

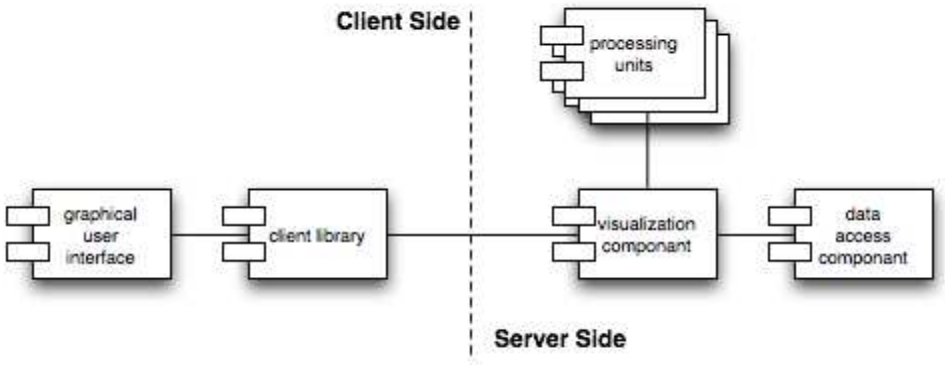
3.2.1 Scenario A: Processing and concept of visualization pipeline

The first proposed step is the modeling of a global SCOS/V3D system as a SCA system. Only the collaborative aspects of the V3D are excluded to focus on the processing aspects. From practical point of view, this scenario attempt to address:

- The service oriented description of a global visualization and complex service;
- The various possibility of client-server binding;
- The facility of development, especially for SCA non-expert;

- The facility of deployment, taking into account for some components of native layers;
- The capability to transfer complex and large dataset between services.

On a first approximation (M1), the system is considered as the composition of a graphical component and a client library on the client side, and of a visualization component, a data access component and a set of components that we call the processing units, on the server side. On this approximation the binding between the two client side components does not have to be done using the SCA specifications. We use SCA on the server side, to describe the components and to bind them to a composite application (i.e. visualization service). This scenario does not take into account the service composition explicitly. This simplified model (M1) is sum-up in the following schema:



The client library communicates with the visualization component thru web services exposed via HTTP. The visualization component consumes both data access component and processing units to render data and to send back 3D scenes. The SCA runtime is typically collocated to the visualization component and binds it to the consumed components.

Processing units can accept parameters and bind each over in order to create complex processes like chains of processes. This implies that we should define them as SCA components and be able to use them through a SCA binding description.

Processing units consume and provide 3D data sets, composed of points, cells defined on those points, and data attached on the points or on the cells. In this approximation, processing units admit a set of scalar values or chars as parameters (or XML setting file).

Because this scenario typically corresponds to the integration of pre-existing applications (i.e. complete 3D viewer, modeling framework, simulation kernel), the purpose of this scenario is also to evaluate the level of difficulty to expose as services such applications for non-experts.

For this reason, the development tools defined in the WP2 have been used.

3.2.2 Scenario B: Use SCA to make the server side a composite application

The aim of the second scenario is to focus on the services composition, especially on the server side. In the scenario each service or “process unit” is associated to a direct processing filter (e.g. cutting plane or iso-surface computation). The main objectives here are:

- Study the effect or the service granularity in terms of
  - Composition facility;
  - Impact on the performances;
  - Data streaming and transfer.
- Deployment issues and testing process of the whole composite application.

- Evaluate the administration and monitoring features.

Using the components developed in the previous step, the final objective is to make a SCA composite application on the server side.

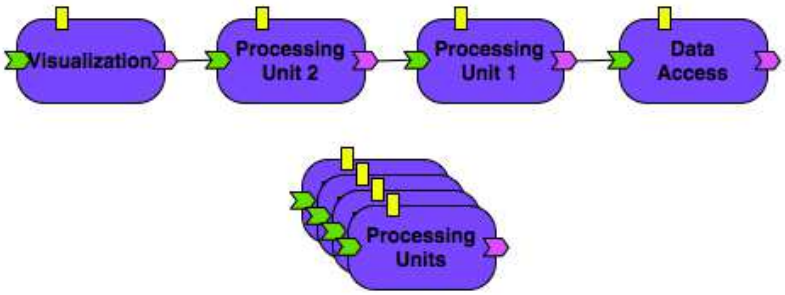


Illustration 3: components and service oriented version of the visualization pipeline

3.2.3 Scenario C: Use semantic to ease process definition

In this step, we will introduce a simple ontology for the concepts we manage in the 3D visualization field, and use it to implement semantic matching between process units.

3.2.4 Scenario D: Create a new processing unit and test it on the client side

We will experiment the creation of a new processing unit on the client side by a scientific worker, and the binding of this new unit into a test workflow.

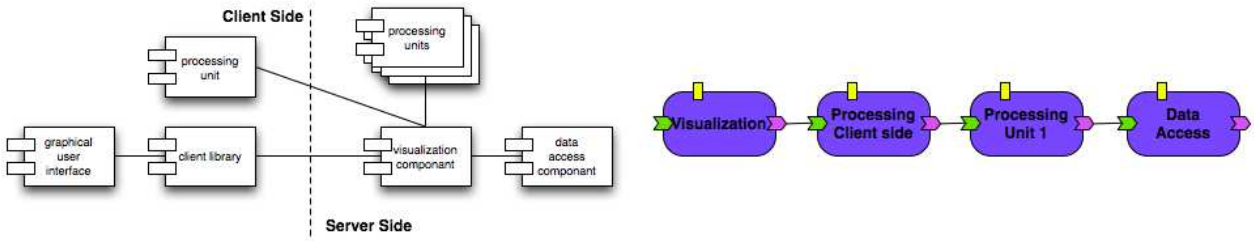


Illustration 4: new component creation

In this step we will create a skeleton of a processing unit and experiment the ease of development and use of a new component for a developer, which is not skilled in component architecture. We will also experiment the versatility of the bindings between the components by using a web services remote access to the client component.

3.2.5 Scenario E: Deploy a new processing unit and test it on the server side

Will experiment the deployment a new processing unit and testing it on the server side. This step should use deployment artifacts developed in the frame of the WP2.

3.2.6 Scenario F: Create an independent application without server connection

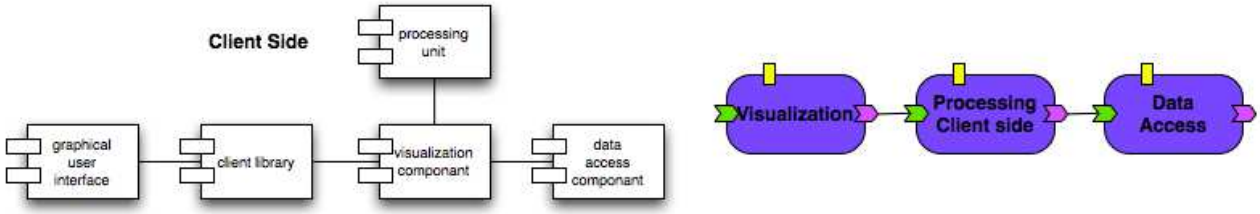


Illustration 5: scientific computing: a standalone client composite application

We will embed the SCA runtime and basic component implementation on the client side to make a composite application without server connexion.

3.3 Implementation and tests

To tests and explore these different scenarios, a set of prototypes have been developed. Some of these are included as code sample in the FraSCAti distribution. However, those software should still considered as proof-of-concept and not be used for operational application purpose.

3.3.1 Demonstrator 1: SCA-CassandraPCS

This first demonstrator focused on the exposition as a service of an existing application and its use with SCA and non-SCA based clients. The goal of this demonstrator is to validate the simplicity to expose with several protocols a single and complex existing component with a total transparency regarding the 3D visualization client consuming this service. This demonstrator typically corresponds to the application scenario A.

The scientific context outlines several key issues.

- Because most of software are pre-existing and used in a classic approach since long time and should maintain the same interfaces in a classic use, it is critical to be able to expose a pre-existing software in a non intrusive manner, i.e. without modification of this one.
- Reciprocally, an increasing number of scientific applications are parallelized using internal high performance MPI techniques, deployed on HPC and exposed as Web service. Another key-issue is then the possibility to access to the exposed service using a standard web service stack.
- Last, the complexity of the exposition procedure is a critical aspect to be adopted by the scientific community and non-SCA specialist.

Practically, this demonstrator was based on the integration of the CassandraPCS VTK based processing engine [5], used in some developments of SCOS/V3D [6]. This integration has been done without modification of the processing engine and the detailed architecture is given in section 4.1 below.

3.3.2 Demonstrator 2: Static data processing

This demonstrator focused on services composition. In the present context, *service composition* should be understood as the building of a system relying in a structured manner to a set of external components. Those external components could be either local or remote. This demonstrator typically corresponds to the application scenarios B, D and E, and more generally to the framework oriented applications.

Here, the concept to service is put at the processing unit or vtkAlgorithm in the VTK vocabulary. In this approach, the visualization pipeline is here directly modeled by a composition of "basic" services. However, in this demonstrator, the dynamical aspect of the service composition has not been treated here, and the visualization process has been modeled in a static manner. In this first step, the focus

was put on:

- Service interoperability: This should be here understood in the sense of "How to replace an implementation of a component by another one". Scientific software generally knows several implementations, based on different numerical models for instance, and addressing the same problem. The critical issue here is the definition of common, standardized and functionally based APIs;
- Study the data streaming and transfer between services;
- Remote control or how to use a remote component instead of using a local one;
- Evaluation of the constraints of using or exposing a components as a remote service;
- Evaluate the performances cost of the local binding with respect a classic implementation;
- Evaluate the complexity of each point.

3.3.3 Demonstrator 3: SCA for complex data processing

In this last demonstrator, the flexibility of SCA based architecture was studied in details and more especially the dynamical services re-composition.

As done in the demonstrator 2, the services are defined here at the level of the processing units (i.e. vtkAlgorithm) and the visualization pipeline directly modeled by composition of services. With respect to the previous experimentation, the focus was put here on the dynamical aspect of the services composition to be closer to realistic cases.

3.4 Architecture / technical design – specification

This section presents the detailed architecture of each demonstrator and their specific technical aspects.

3.4.1 Demonstrator 1: SCA-CassandraPCS

As presented in the section above, this demonstrator is based on the exposition of a global visualization service. This service is based on CassandraPCS, which is a 3D data processing engine written in Java and based on VTK. According to the concept of visualization pipeline of VTK, the principle of Cassandra is to process the scientific data by combination of a set of basic filters (i.e. vtkAlgorithm). Scientific dataset are given in input and processed along this pipeline. In output, CassandraPCS generates a 3D reality scene corresponding to the result of the processing (e.g iso-surface), under the form a X3D/VRML file. The control of the processing engine is done through a set of commands gathered in a single XML query. The schema in Illustration 6 illustrates this approach.

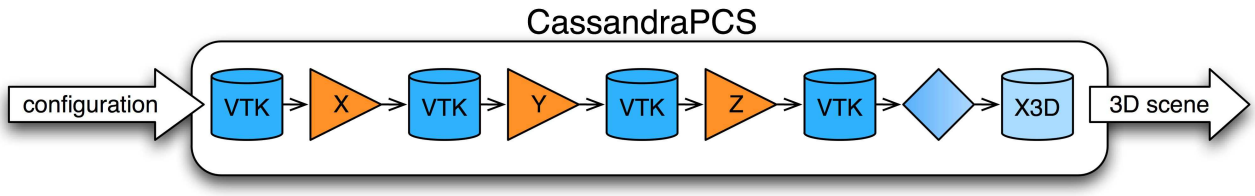


Illustration 6: service exposition of the CassandraPCS processing engine

This engine can be piloted by an external end-user application, in client-server mode for instance. The operation of the CassandraPCS processing engine is functionally de facto close to a service approach, with input data processed on demand according a given setting (configuration file).

However, a few small adaptations were needed with respect to the initial design of CassandraPCS.

First, SCA requires an interface for the service exposition and a concrete class for its implementation

and in its standard distribution, CassandraPCS did not provide any interface. In consequences, a new interface and a new implementation were made for the service exposition, providing by the way an abstraction layer. Second, the standard implementation of CassandraPCS is statefull, which means that it own an internal state for progressive data processing definition, which could be achieved by several method's calls. To deploy the engine in a SCA context, a new implementation has been developed, providing a stateless service by using the CassandraPCS service as local variable inside the processing method. Illustration 7 gives a graphical representation of the SCA domain of this service.

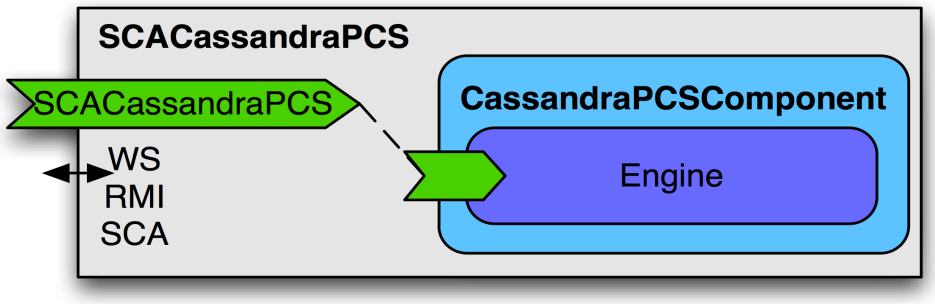


Illustration 7: service oriented architecture of CassandraPCS

The SCA runtime implicitly offers several bindings. In order to load a SCA service into the runtime, an XML file must be read and processed by it. Once the processing of this file is done, all the services and components attached to it become available. The SCA domain is given below as a XML composite file.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<composite name="SCACassandraPCS"  xmlns="http://www.osoa.org/xmlns/sca/1.0" >

  <service name="SCACassandraPCS" promote="CassandraPCS/Engine">
    <interface.java interface="com.artenum.scorware.cassandra.Engine"/>
  </service>

  <component name="CassandraPCS">
    <implementation.java class="com.artenum.scorware.cassandra.CassandraPCSEngineImpl"/>
    <service name="Engine">
      <interface.java interface="com.artenum.scorware.cassandra.Engine"/>
      <binding.ws uri="http://192.168.0.11:8080/CassandraPCS" />
      <binding.rmi host="localhost" serviceName="rmiservice" port="1099"/>
    </service>
  </component>
</composite>
```

Composite definition of complex service exposition

To achieve the deployment, an execution class as been done in order to load the service inside the runtime. A very simple example of the launching application to deploy the service on the server is given below.

```
public static void main(String[] argc) throws Exception {
        // Load the SCA domain by reading the SCACassandraPCS.composite XML file.
        Component scaDomain = AssemblyFactory.getComposite("SCACassandraPCS");
        // Generate an instance of the SCACassandraPCS service managed by the SCA runtime
        Engine engine = TinfiDomain.getService(scaDomain, Engine.class, "SCACassandraPCS");
        // Once the service instance is available, the service is exposed.

        // Wait a user input in order to stop the service
        System.in.read();

        // Stop the service
        TinfiDomain.close(scaDomain);
}
```

Main class for deploying the service

Two dedicated clients have been developed, one based on SCA binding and one based on a default Web Service stack, in order to test the different ways to access and use the service.

The user interface of these clients is composed of a 3D view, to display the resulting scene generated

by the service, and a query panel, to control the service. The control of the service is done by editing a set of commands gathered in a XML based file. The client application simply calls the service and display the 3D data generated by the service to the user. A view of the client GUI that client is given in Illustration 8 below.

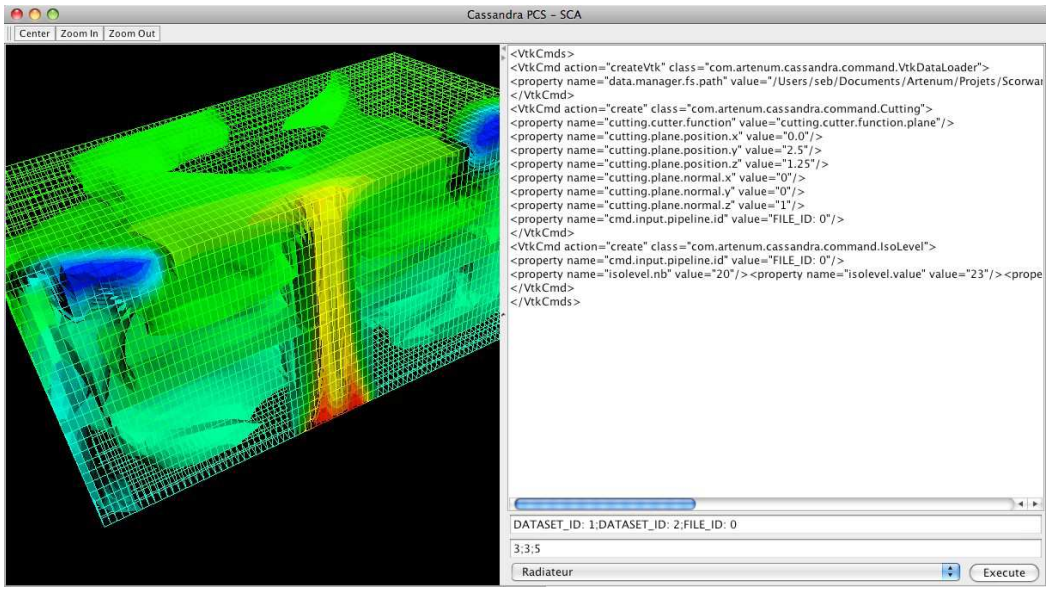


Illustration 8: view of the CassandraPCS client

Most of scientific applications have to be deployed in heterogeneous context, where the potential clients may access to the services in different manners and using different bindings. From this point of view, SCA presents potentially a strong advantage with respect to other solutions. Both listings for the SCA and Web Services binding are given below. These main classes show the injection of the service, instantiation and the initialization of the client and, last, the service closing procedure.

```
public static void main(String[] argc) throws Exception {
        CassandraPCS apps = new CassandraPCS();
        // Injection of the engine
        Component scaDomain = AssemblyFactory.getComposite("WSClientCassandraPCS");
        Engine engine = TinfiDomain.getService(scaDomain, Engine.class, "SCACassandraPCS");
        apps.setEngine(engine);
        // Build the Frame of the application
        JFrame f = new JFrame("Cassandra PCS - SCA");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(500, 500);
        f.getContentPane().setLayout(new BorderLayout());
        f.getContentPane().add(apps, BorderLayout.CENTER);
        f.setVisible(true);
        // Initialise the graphical client
        apps.init();
        // Wait user input to close the generated Engine
        System.in.read();
        // Close the SCA domain
        TinfiDomain.close(scaDomain);
}
```

SCA based client

```
public static void main(String[] argc) throws Exception {
        CassandraPCS apps = new CassandraPCS();
        // Injection of the engine
        ClientProxyFactoryBean factory = new ClientProxyFactoryBean();
        factory.setServiceClass(Engine.class);
        factory.setAddress("http://88.182.100.19:9000/CassandraPCS");
        apps.setEngine((Engine) factory.create());
        // Build the Frame of the application
        JFrame f = new JFrame("Cassandra PCS - WS");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(500, 500);
        f.getContentPane().setLayout(new BorderLayout());
        f.getContentPane().add(apps, BorderLayout.CENTER);
        f.setVisible(true);
        // Initialise the graphical client
        apps.init();
}
```

Standard WebService based client

The last test was to access to the service from a pre-existing and not specific client. This was done through the SPIS-UI system [7], based on the Kerridwen IME, by loading a Web Service stack library and using it dynamically by executing a Python/Jython script. Illustration 9 illustrates the SPIS-UI GUI, the messages log in Jython console and the results displayed in the integrated viewer.

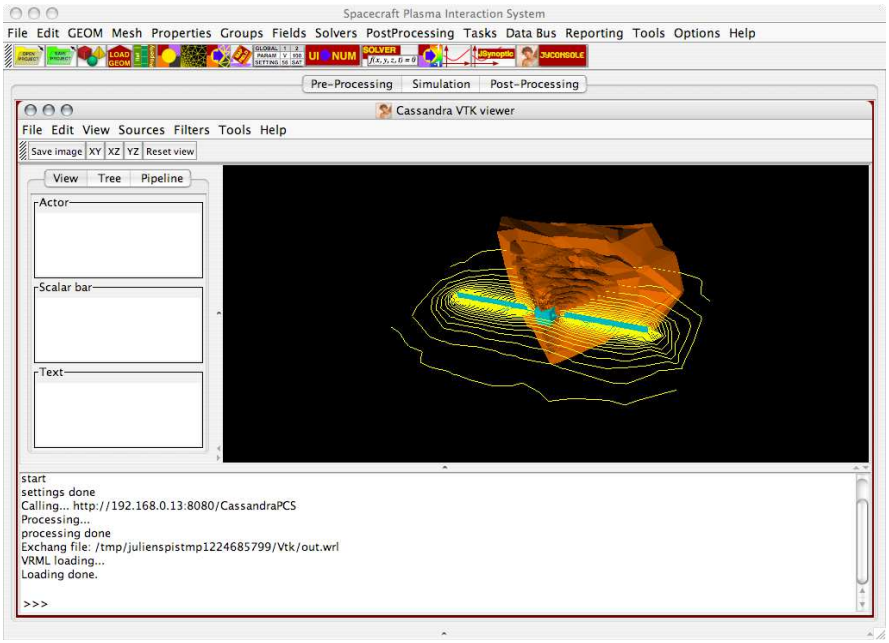


Illustration 9: view of the use of the visualization service from the pre-existing SPIS application

In this approach the SCA binding is not used. The purpose of this test is mainly to check the facility of use of the service by another application and using a high-level script language. The corresponding script is given below. In the present case, the control commands are previously defined in a simple Python dictionary. The service is then invoked and processed, and the output data are recovered under the form of a VRML file. This last one is then loaded and displayed in Cassandra 2.0, in a classic manner.

```
from org.apache.cxf.frontend import ClientProxyFactoryBean
factory = ClientProxyFactoryBean()
factory.setServiceClass( Engine )

hostAdress = javax.swing.JOptionPane.showInputDialog("Server host address")
factory.setAddress("http://"+hostAdress+":9000/CassandraPCS")
engine = factory.create()

data = engine.process(commande, fields, rep)

vrmlTmpFile = GL_VTK_EXCHANGE+os.sep+"out.wrl"
out = file(vrmlTmpFile, "w")
out.write( data.tostring())
out.close()

cassandra = Cassandra()
cassandra.getRendererPanel().GetRenderer().RemoveAllProps()

importer = vtkVRMLImporter()
importer.SetFileName(vrmlTmpFile)
importer.SetRenderWindow( cassandra.getPipeLineManager().getCassandraView().GetRenderWindow())
importer.Update()
```

Integration of CassandraPCS service inside SPIS framework

3.4.2 Demonstrator 2: Static data processing

In this demonstrator, a component wrapping has been done at the level of the filters themselves (vtkAlgorithm) as *Process Units* that are combined into a pre configured processing chain, defined as

a static service composition in a *DefaultPipelineComponent*.

This DefaultPipelineComponent is the only one to be exposed, either remotely or locally, and handled by the client applications. Moreover, the Process Units composing the pipeline can themselves be deployed on the same domain (i.e. computer) than the DefaultPipelineComponent or remotely (i.e. another computers). Illustration 10 illustrates this configuration.

In such design, the original data models may introduce difficulties in the data sharing between processing units. Such models are generally linked to pre-existing and tailored algorithms that do not take into account the requirements of distributed designs and more generally SOA. For instance most of historical scientific data formats does not support the serialization of data, needed to transfer them through streams.

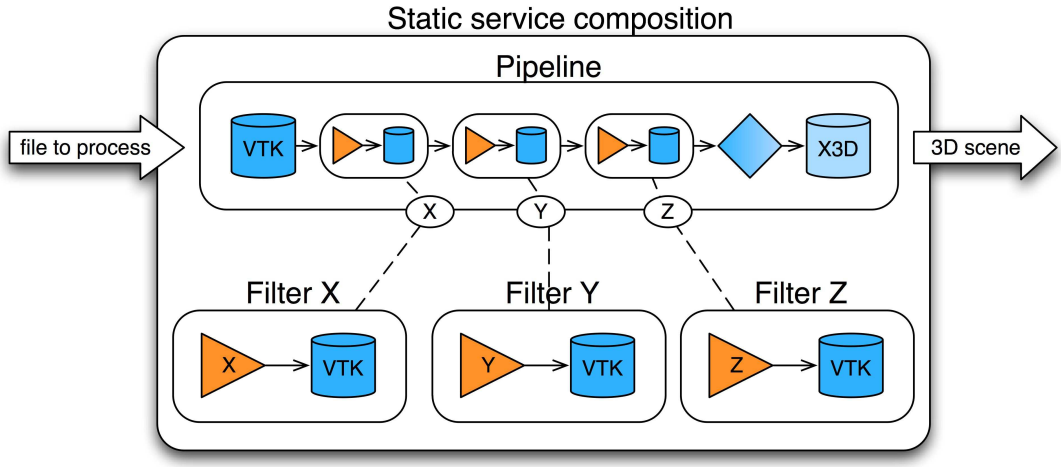


Illustration 10: services composition

This is the case of vtkDataSet, used here to store the scientific data, which are written in C++ and for which ones the corresponding Java object do not implement the *Serialisable* interface. If here the difficulty is mainly due to technological reasons, one should outline that similar problems may be due to the data structure themselves, if this one presents a cyclic structure for instance or if the data are distributed on a set of sub-sets with cross references.

These aspects have been studied by two different implementations of the demonstrators.

In the first one, the focus was put on the service composition, a local deployment and the direct manipulation of wrapped native objects, including not serialisable ones, by the processing units. The graphical representation of the SCA domain is given in the following graphic.

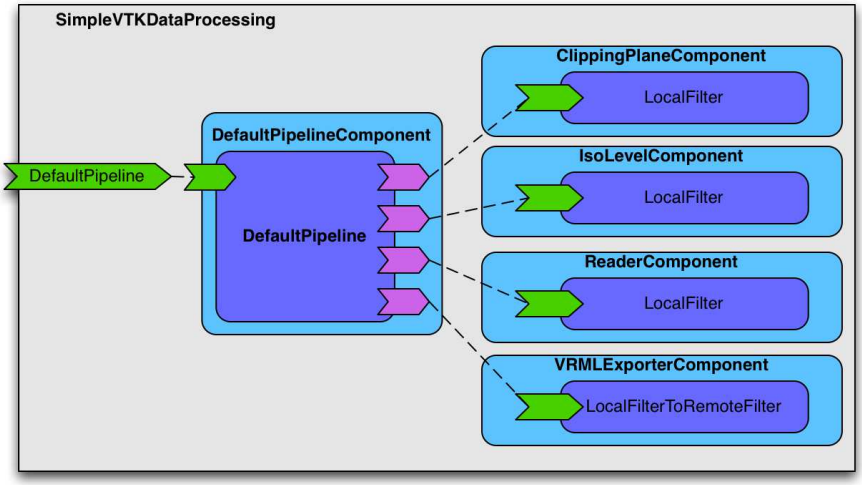


Illustration 11: static processing service based on local processing units

In practice in the present demonstrator, the processing chain has been hard coded and the configuration of the processing service is static. The final exposed API is simplified in consequences. The following extract of listing illustrates this configuration.

```
public class DefaultPipeline implements RemoteFilter {
      private LocalFilter clip;
      private LocalFilter iso;
      private LocalFilter reader;
      private LocalToRemoteFilter exporter;

      @Reference(name = "clip")
      public void setClip( LocalFilter clip ) {
         this.clip = clip;
      }

      @Reference(name = "iso")
      public void setIso( LocalFilter iso ) {
         this.iso = iso;
      }

      @Reference(name = "reader")
      public void setReader( LocalFilter reader ) {
         this.reader = reader;
      }

      @Reference(name = "exporter")
      public void setExporter( LocalToRemoteFilter exporter ) {
         this.exporter = exporter;
      }

      public byte[] process( Properties properties , byte[] data ) throws Exception {
         Properties clip1 = new Properties();
         clip1.setProperty(ClipingPlaneKeys.NORMAL_X, "0");
         clip1.setProperty(ClipingPlaneKeys.NORMAL_Y, "1");
         clip1.setProperty(ClipingPlaneKeys.NORMAL_Z, "1");
         clip1.setProperty(ClipingPlaneKeys.ORIGINE_X, "0");
         clip1.setProperty(ClipingPlaneKeys.ORIGINE_Y, "-0.40");
         clip1.setProperty(ClipingPlaneKeys.ORIGINE_Z, "0");

         Properties clip2 = new Properties();
         clip2.setProperty(ClipingPlaneKeys.NORMAL_X, "0");
         clip2.setProperty(ClipingPlaneKeys.NORMAL_Y, "-1");
         clip2.setProperty(ClipingPlaneKeys.NORMAL_Z, "1");
         clip2.setProperty(ClipingPlaneKeys.ORIGINE_X, "0");
         clip2.setProperty(ClipingPlaneKeys.ORIGINE_Y, "0.40");
         clip2.setProperty(ClipingPlaneKeys.ORIGINE_Z, "0");

         double[] scalarRange = { -0.10792893916368484, 0.0016820
         Properties isoProp = new Properties();
         isoProp.setProperty(IsoLevelKeys.ISO_LEVEL_VALUES, "-0.1:-0.09:-0.08:…:0");

         vtkDataSet ds = null;
         long startingTime = System.currentTimeMillis();
         ds = reader.process(properties, ds);
         ds = clip.process(clip1, ds);
         ds = clip.process(clip2, ds);
         ds = iso.process(isoProp, ds);
```

```
        byte[] result = exporter.process(null, ds);
        System.out.println("Processing time: " + (System.currentTimeMillis() - startingTime));
        return result;
    }
```

The next development was based on the same principle but try to use remote processing units inside the global processing service. To address this configuration, adaptations were needed to provide serialisable parameters. Regarding the vtkDataSet this has been done by the creation of temporary structures that are simply converted into byte arrays. Thanks to this and the components abstraction, Process Units have been deployed both locally and remotely, as illustrated in Illustration 12.

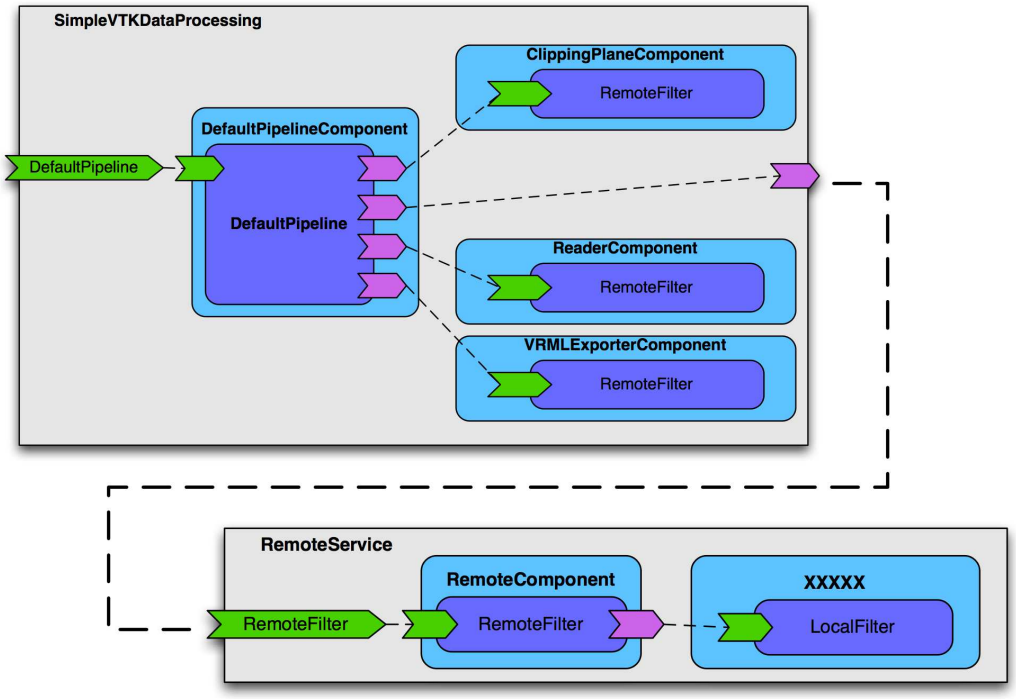


Illustration 12: static processing service based on a set of processing units with one remotely localized

The files listed below respectively correspond to the SCA domains in local and remote deployment of process units. The comparison of both listings shows that, after a proper definition of services, how is simple to change from one configuration to another one. Such conclusion may highly useful in scientific applications, where the scalability of the whole application is frequently a key factor. The services distribution may also answer to an operational need in some project, where data should be first pre-processed remotely before any transfer, to reduce their size or complexity, for instance.

```
...
<component name="DefaultPipeline">
  <implementation.java class="com.artenum.scorware.vtk.pipeline.DefaultPipelineRemote"/>
  <service name="RemoteFilter">
    <interface.java interface="com.artenum.scorware.vtk.processing.RemoteFilter"/>
  </service>
  <reference name="clip" target="ClippingPlaneComponent/RemoteFilter">
    <interface.java interface="com.artenum.scorware.vtk.processing.RemoteFilter"/>
  </reference>
  <reference name="iso" target="IsoLevelComponent/RemoteFilter">
    <interface.java interface="com.artenum.scorware.vtk.processing.RemoteFilter"/>
  </reference>
  <reference name="reader" target="ReaderComponent/RemoteFilter">
    <interface.java interface="com.artenum.scorware.vtk.processing.RemoteFilter"/>
  </reference>
  <reference name="exporter" target="VRMLExporterComponent/RemoteFilter">
    <interface.java interface="com.artenum.scorware.vtk.processing.RemoteFilter"/>
  </reference>
</component>
...
```

Local components with remote compliant API

```
...
<component name="DefaultPipeline">
   <implementation.java class="com.artenum.scorware.vtk.pipeline.DefaultPipelineRemote"/>
   <service name="RemoteFilter">
     <interface.java interface="com.artenum.scorware.vtk.processing.RemoteFilter"/>
   </service>
   <reference name="clip" target="ClippingPlaneComponent/RemoteFilter">
     <interface.java interface="com.artenum.scorware.vtk.processing.RemoteFilter"/>
     <binding.ws uri="http://192.168.0.10:8080/remoteClip" /> <!-- CHANGE HERE !!! -->
   </reference>
   <reference name="iso" target="IsoLevelComponent/RemoteFilter">
     <interface.java interface="com.artenum.scorware.vtk.processing.RemoteFilter"/>
   </reference>
   <reference name="reader" target="ReaderComponent/RemoteFilter">
     <interface.java interface="com.artenum.scorware.vtk.processing.RemoteFilter"/>
   </reference>
   <reference name="exporter" target="VRMLExporterComponent/RemoteFilter">
     <interface.java interface="com.artenum.scorware.vtk.processing.RemoteFilter"/>
   </reference>
</component>
...
```

Use of a remote component for the clip filter

These files also define the injection rules and the selection of the used implementation. As example, the following extract of listing illustrates how to switch between several implementations. This could be simply achieved by changing the class name of a component. Such functionality may be very useful to change of numerical models, for instance, in a same processing chain.

```
...
<component name="IsoLevelComponent">
   <implementation.java class="com.artenum.scorware.vtk.processing.filter.remote.XXXXXXX"/>
   <service name="RemoteFilter">
     <interface.java interface="com.artenum.scorware.vtk.processing.RemoteFilter"/>
   </service>
</component>
...
```

3.4.3 Demonstrator 3: SOA for complex data processing

The final demonstrator tries to highlight the most advanced features of SCA runtimes, by providing a flexible service exposition to a complex and dynamic data processing service. By flexibility, we mean a processing service that can be extended by any number of processing units, which could be indifferently deployed locally or remotely.

Classically, a main set of processing units is defined in the SCA domain before the runtime.

However, the design of FraSCAti allows to dynamically add any extra processing unit to the SCA domain at the runtime, with some specific implementations. This extensibility has only been done for remote RMI processing units, but could be as easily extended to any binding.

In order to be closer to the architecture of operational projects like SCOS/V3D, the process units were distributed on several remote nodes and inter-connect using the RMI and WebService binding.

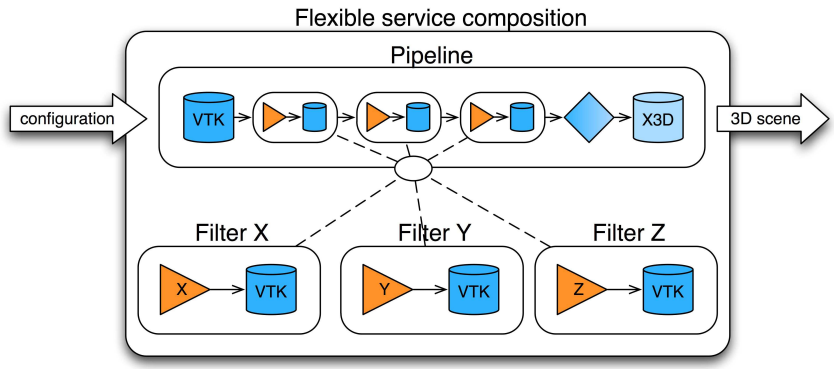


Illustration 13: dynamical service composition

The SCA domain used to describe this application is provided in a figure below. The component

named PipelineComponent is responsible of the configuration and the composition of each registered processing units.

The process units can then be linked to the PipelineComponent. This last one will offer the way to manage them through a single injection point, which allows any number of processing units.

Nominally, the SCA specification does not offer the possibility of dynamical service compositing. The service compositing is done through the definition of the SCA domain and requires a re-starting of the whole system, if the services connectivity has been modified. But, a dynamical service composition corresponds to a critical need in most scientific applications. For instance, it appears as prohibitive to have to restart a complete visualization system for each modification of the visualization pipeline.

However, this difficulty has been handled in the demonstrator 3 by the following approach. In a first step, the Process Units are registered and referenced into the PipelineComponent using the SCA domain, as usual. By this, the PipelineComponent offers a central reference to each ProcessUnit.

The composition itself, i.e. the connectivity between the Process Units, is not directly expressed in the SCA domain, but using a dedicated tree managing the services connectivity defined in the PipelineComponent. This structure can dynamically be defined and modified. Each Process Units is then processed in the correct ordering, as defined in the PipelineComponent. It should be noticed that, due to the progressive pipeline definition, the PipelineComponent should be statefull, which can be easily done by declaring this one with a “Composite” scope in the code annotation.

A last point should be outlined. If the services declaration through the SCA domain is normally done at the runtime initialization with most of other SCA implementations, FraSCAti offers an additional API than allows to dynamically declaring new components without need to restart the runtime.

Regarding the user experience, this approach presents the strong advantage to prevent any code edition, nor compilation to extend the global service. This approach allows a complete and dynamical way to extend the visualization pipeline. A processing unit definition is shown as example in the SCA composite file below.

```
…
<component name="IsoContourFilterComponent">
        <implementation.java class="com.artenum.scorware.vtk.filter.IsoContourFilter" />
        <service name="Filter">
            <interface.java interface="com.artenum.scorware.vtk.api.LocalFilter" />
        </service>
</component>
<wire source="PipelineComponent/filter3" target="IsoContourFilterComponent/Filter" />
…
```

The SCA domain illustrating the global architecture of the application is given below.

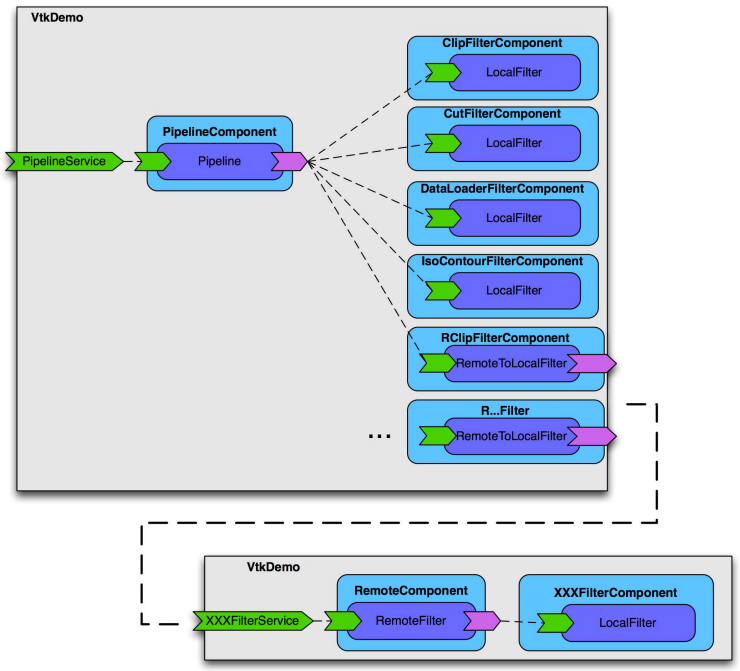


Illustration 14: the figure illustrates both a remote processing unit domain and the global processing service domain

Two implementations of the PipelineComponent have been done, one generating a vtkDataSet and another one generating a standard VRML file. A client application has been developed for each version of the service, using different technologies of 3D rendering, depending on the nature of the produced data. Screenshots of both clients are given in Illustration 15 and Illustration 16.

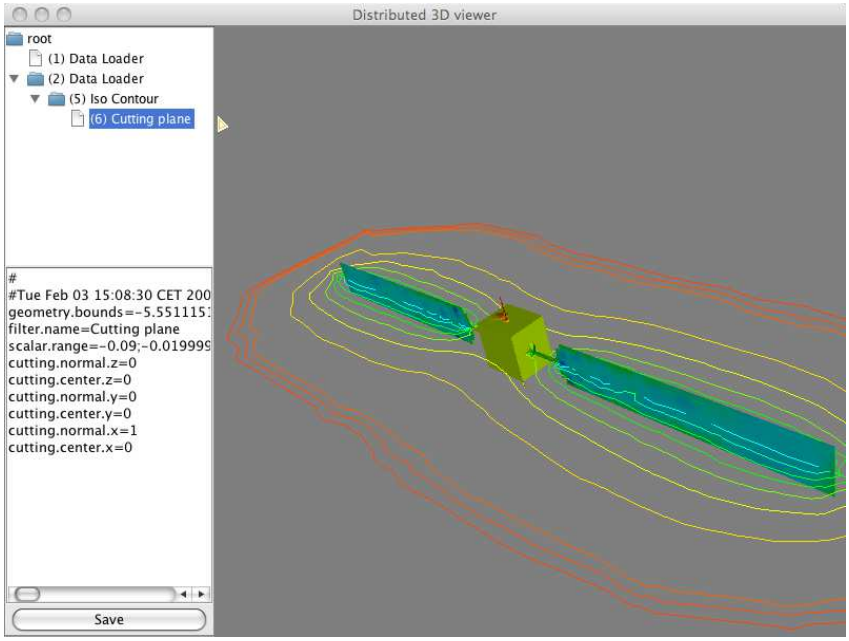


Illustration 15: client based on the Java 3D renderer

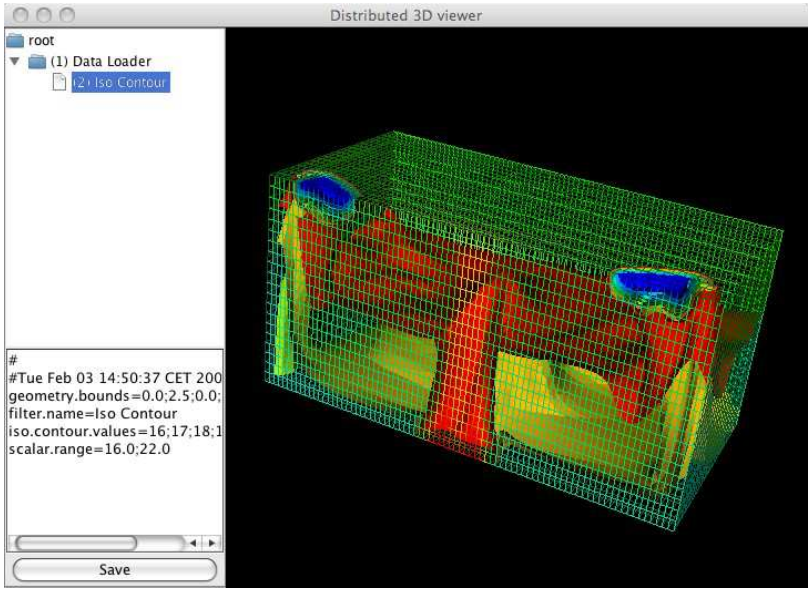


Illustration 16: client based on the VTK renderer

3.5 Coverage of the demonstrator (SCA spec, SCOrWare technical platform)

In the frame of the WP 3.1, it was decided to support an approach focusing on application cases the closest possible to realistic situation in scientific computing and data analysis. The scenarios/ demonstrators compliance matrix below summarizes the coverage of the proposed scenarios by the different demonstrators. By the way, the results may seem global or integrative and any demonstrator can be explicitly related of a specific feature of SCA or another one. However, thanks to the riches and the complexity of these scenarios a large set of functionalities is de facto covered:

- ✓ Local and remote deployment;
- ✓ Use of different binding (SCA, RMI, Web services…);
- ✓ Integration and deployment of native components through a JNI wrapping;
- ✓ Transfer of large and complex data set and identification of related constraints (i.e. serialisable structures);
- ✓ Modeling of service composition, statically and dynamically;
- ✓ Exposition of stateless and state full services.

The use of abstract semantics was unfortunately not explored in the frame of this study, in spite its large potential field of application in scientific applications.

	Demonstrator 1	Demonstrator 2	Demonstrator 3
Scenario A	X	X	
Scenario B		X	X
Scenario C			
Scenario D		X	X
Scenario E			X
Scenario F			X

Table 2: SCA and SCOrWare coverage (scientific computing)

3.6 Lessons learned / methodology / best practices / demonstrators' results

The demonstrators developed in the frame of the WP 3.1 focused on applications cases related to scientific applications and, more precisely, 3D data analysis. A step-by-step approach was followed in order to study the various functionalities offered by SCA. A first step was focused on the service exposition, with the use of several bindings and different deployment contexts. In a second phase, a fully SOA based architecture and the services composition was studied in more details, with the conversion of the canonical visualization pipeline into a composition of services.

From a human point view, the followed methodology was voluntary based on practical cases in order to evaluate the learning curves of SCA especially for non-experts. Indeed, such social and cultural aspects may be critical for a success dissemination of a new software technology. The development of these demonstrators has confirmed that the introduction of SCA does not induce a prohibitive overhead, if the migration toward a SOA is performed step-by-step. These tests have also confirmed that SCA answers, in a large part, to the needs of the scientific community in this field.

However, these first tests have shown that a full migration to service oriented architecture may require some design and paradigm modification in existing software. For instance, the distribution of services on different nodes with SCA forbids the sharing of data, through references, and requires cloning them while remote communication is involved. Therefore, it becomes impossible to share a reference between services in a remote context. SCA does not offer an automatic proxy to manage call-back between distributed objects by just wiring calls among services to the real object instance. This is at the opposite of ProActive based techniques [8], with Active Object when they are used by several remote distributed components. This should be taken into account into the global software design.

This specific point is not a drawback of the SCA technology. But, what is important to understand is that SCA do not manage the same way its local and remote components.

With the distance, one key point for distributed high performance computing is probably to identify clearly the right service granularity, depending on the nature of the service, the deployment and network constraints. For instance, the use of SCA of service exposition of coarse components, like in demonstrator 1, seems well adapted. The various binding possibilities offered by SCA is here a strong advantage. The same observation can be done regarding its facility of configuration and deployment, thanks to the definition of the SCA domain. This does not forbid the use of MPI or ProActive approaches for the fine components distribution in Intranet on HPC. Both techniques seem complementary of each other.

The whole performances evaluation was initially a source of worrying, with the evaluation of the over-cost due to the service exposition. Tests performed, comparing an in-lined version of the processing chain and the SCA based version, have shown that the difference seems not significant, at least for applications similar to the demonstrator 1.

The main feedbacks collected to this first experimentation are finally:

- Easy service injection in an independent manner of the service exposition like Spring [9].
- Small impact on code change while exposing a service. (Annotation needed for scope and references...) The pattern adapter can be used to prevent code intrusion in existing software where source code can't be changed or annotated.
- The service exposition has been simplified in a drastic manner specially when the same service is exposed with several protocols.

The developed demonstrators should still be considered as simple proof-of-concept and it is not recommended to use them for operational applications. The demonstrators 1 and 2 are structured as Maven based projects and are available on demand. The third demonstrator, illustrating the dynamically service composition, is packaged under the form of an Ant based project and is including

in the FraSCAti runtime samples (repository SVN). Released under the terms of the LGPL license, it can be freely used, modified and extended.

3.7 References

[1] Salome plate-forme Web site, www.salome-plaform.org

[2] VTK Web site, www.vtk.org

[3] OpenDX Web site, http://www.opendx.org

[4] S. Jourdain et. al., ShareX3D, a scientific collaborative 3D viewer over HTTP, Web3D Symposium 2008, Los Angeles, California, 2008.

[5] Cassandra's Web page, www.artenum.com/cassandra

[6] SCOS project Web page, www.oscos.org

[7] SPINE community Web site, www.spis.org

[8] ProActive Web site, http://proactive.inria.fr/

[9] Spring Web site, http://www.springsource.org/

4 COLLABORATIVE DEVELOPMENT (Task 3.2)

Change history vs. version 1.0: the first version 1.0 (October, 2007) of the specification was describing the first stage of the development of this usage demonstrator (with Tuscany as target runtime SCA platform), focusing on the functionalities of a New Generation SOA Development Forge. This final version specifies the integration of a Collaborative Environment, through the so-called OS2P Portal (Open Source Project Portal). It considers too the replacement of the former portal technology foreseen in this usage demonstrator (i.e. Exo Portal and WebOS) by a new one (i.e. Liferay), following the withdraw of the partner eXo Platform during the first year of the project.

4.1 System context

4.1.1 Forge domain model

The domain of CDE inherits from collaborative environment (computer supported collaborative work, or CSCW) and from software development environment (software engineering) domains.

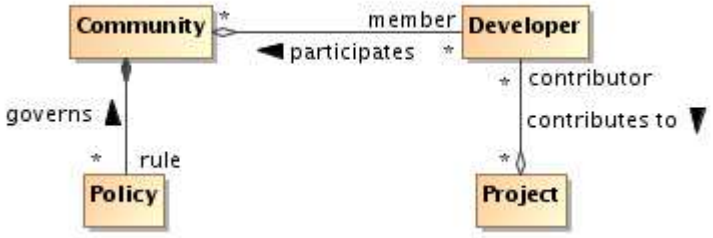


Illustration 17: Collaborative development: forge domain model

The collaborative aspect of the domain defines a community as an aggregation of individual members achieving collaboration through participation according a set of common rules. The development environment aspect of the domain defines members as software developers that concurrently and collaboratively contribute to the development of a software project.

A forge is a configured collaborative platform hosting the software development projects of a community of developers.

4.1.2 Project portal domain model

Software projects are used as foundation or building blocks by software integrators to develop business- and client- specific solutions. However, in order to do this, software projects are not merely put side to side, but integrated together through configuration files, technical "glue" development, and made to interact using business specific model and logic.

We define the reusable part of this work as a "project integration" for a given use case or business need. Another way to see it is 1. that a project integration is in a “uses” relation with the project(s) it integrates and 2. that it is not a “top-level” project, top-level meaning here that it is not meant to appeal on its own and outside the scope of the project(s) it integrates.

We also define the largest context of a project portal, which:

1. provides a configured collaborative platform hosting project integrations provided by the community of software integrators,
2. 2. provides a web publishing solution targeting project users and other software integrators.

4.1.3 Business model

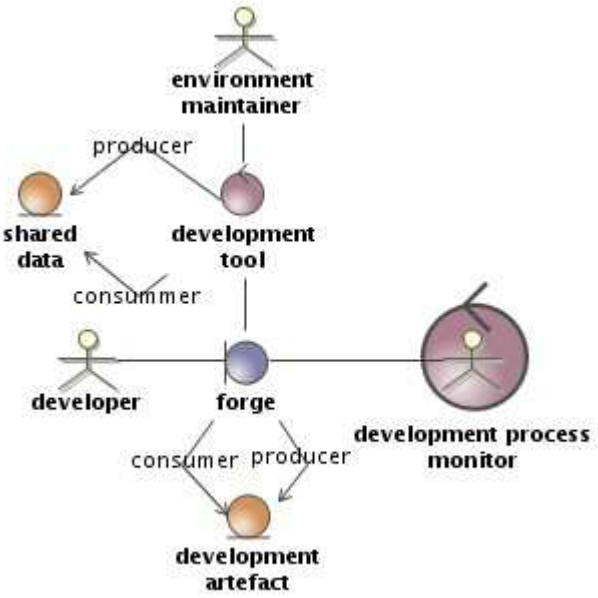


Illustration 18: Collaborative development: business model

The forge is the interface that implements collaborative development business.

The system shall ease the development and deployment of development oriented services, participating in the collaborative development workflow.

The system shall enable automation of the development process monitoring.

The project portal shall allow project integrators to manage documented use cases, as well as related features and project integrations, and make information about those available to potential project users.

Additional roles needed in the project portal solution are : project integrator and technology user (which may be as well a potential software integrator as a business analyst).

4.1.4 Application domain

The application domain was specified in the presentation "Open Forge Standards – Basic Specifications – FP6 QualiPSo project – Olivier Abdoun, INRIA August 2007", delivered with the first version 1.0 of the specification (2007, October).

4.1.5 Project portal: CMS features in a forge

4.1.5.1 Context

Open Wide is a software solution integrator and as such collaborative development tools and middleware such as a "forge" is one of the primary tools used by its development teams to be able to efficiently develop projects answering its clients' needs.

Being at the same time not only a specialist in Open Source technologies, but also a practicing advocate of Open Source methodologies, Open Wide has encountered the need to expand the use of such forges to new features in order to be able to target a greater population, comprising not only developers but also "clients" i.e. business oriented users, as well as fostering use and integration of said project.

- project communication management and publication
- project evolution, feature requirements and roadmap
- project use cases management
- project integrations management

We name such a project portal applied to open source projects Open Source Project Portal (OS2P). In this vision, the forge functionalities are integrated into the OS2P portal, and forge tools are accessible though portlets.

In order to answer these requirements, we can rely on the proposed architecture, with an emphasis on document management, publication and knowledge management.

4.1.5.2 Project communication management and publication

Its point is to literally make people other than developers welcome on the site. This area will also be a help to integrators trying to sell solutions integrating this Open Source project, by providing client-oriented communication information.

This is achieved by a portal-integrated CMS (web content management system). The point is to provide CMS features integrated to OS2P, including document publication.

4.1.5.3 Project evolution, feature requirements and roadmap

Its point is to foster and aggregate project vision and evolution discussions, user suggestions, integrator feature requirements as much as possible in order to be able to easily check roadmap consistency, but as well to track causes of forks and, at a lesser level, of externally developed extensions, and why not possible concurrency or partner projects.

It has been studied by Open Wide in the first year. However, since it is a rather well-known field of Open Source project web-based forges such as GForge, it has not been implemented in the final release.

4.1.5.4 Project use cases management

Use cases are not what Open Source project are most well known for. However it's where a potential user will most easily recognize its own problematic, and therefore will be interested in a given project.

This is achieved by a Knowledge Management-like web application, linkedwith the below mentioned integrations (technology stack) management .

4.1.5.5 Project integrations (technology stack) management

Open Source projects almost alwaysneed some kind of technical integration to become useful. The work to be done here is neither big nor very difficult, but the knowledge required is usually buried in FAQ, forums and other community sections - while its source code is most of the time not even SCM managed.

This is done by managing projects, their "integrates/uses" relationships, and their top-levelness.

Managing technology stacks that build on this project will be achieved through building on the features of the entire demonstrator platform.

4.2 Usage scenarios

4.2.1 Forge #1: new source code revision quality check

A developer has modified a project's source code in his local workspace. He wants to integrate the patch to the forge Source Code Manager. The SCM is configured to verify that the source tree respects coding standards specified by a policy.

Main sequence

1. the developer submits the patch to the forge SCM
2. the forge evaluates the quality of the submission
3. the evaluation succeeds
4. the forge publishes the evaluation result to content management
5. the forge notifies the developers that a new revision is available

Exceptional sequence

1. the evaluation fails
2. the forge publishes the evaluation to content management
3. the forge notifies the developers of the patch integration failure

4.2.2 Forge #2: development project road map

This scenario provides simple project road map features, and involves two different stakeholders:

- the project manager, who wants to define the road map and have a progress overview of the project
- the developers, who want to know which tasks have to done and completed

Dashboard model

- project presentation: name, description
- checkpoint identification: milestone
- a milestone is defined by a list of tasks and a deadline

Main sequence

1. the project manager defines the road map
2. the developer looks at the tasks to be completed

3. the developer sets the task progress

Events

- when a task is completed (notified by the developer)
- when a milestone date is up (notified by the road map service)

4.2.3 Project portal #1: managing use cases, features and project integrations

Technology users : Adding a proposed use case

1. the technology user describes the proposed use case
2. the technology user optionally links it to related features or requested features
3. the project portal manager approves or disproves the proposed use case

Project integrators : Adding a project or a project integration

1. the project integrator describes and provides the project and sets its top-levelness
2. the project integrator optionally links it to integrated projects
3. the project integrator optionally links it to related use cases
4. the project integrator optionally links it to related features
5. the project integration goes through an approval workflow within the community

4.2.4 Project portal #2: CMS features

Technology users : Publishing project content

1. the technology user provides the document to be published
2. it is optionally set to approval (workflow) by the project integrator
3. the published content is available to download by everybody

4.3 Design

4.3.1 Forge components

The application is split up into components and 3 layers:

- the service layer that serves forge functionalities to users in a uniform way and integrates the business components in a coherent way
- the business layer that implements the different functional areas of collaborative software development
- the collaboration support layer that implements human on line collaboration features.

4.3.1.1 Global view

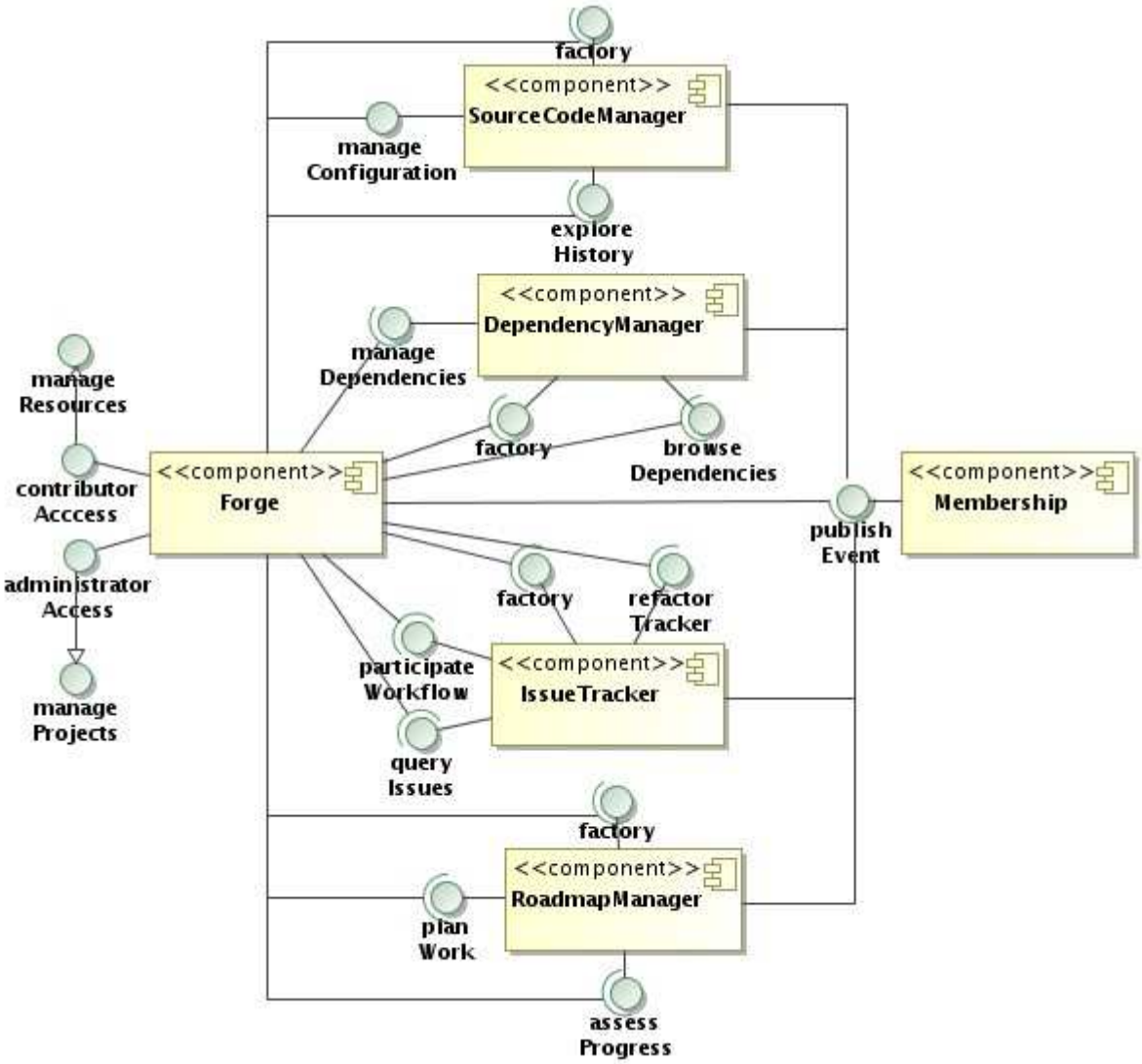


Illustration 19: Collaborative development: forge components (global view)

The Forge component implements the setup of collaboration space for software project development (administratorAccess interface) and the concurrent and collaborative production of software projects artifacts (contributorAccess interface). This is realized through usage and promotion of the interfaces of the s*oftware development components* (SourceCodeManager, DependencyManager, IssueTracker and RoadmapMananger). The Membership component provides the team building and awareness functionalities of a collaborative framework.

4.3.1.2 Implementation view

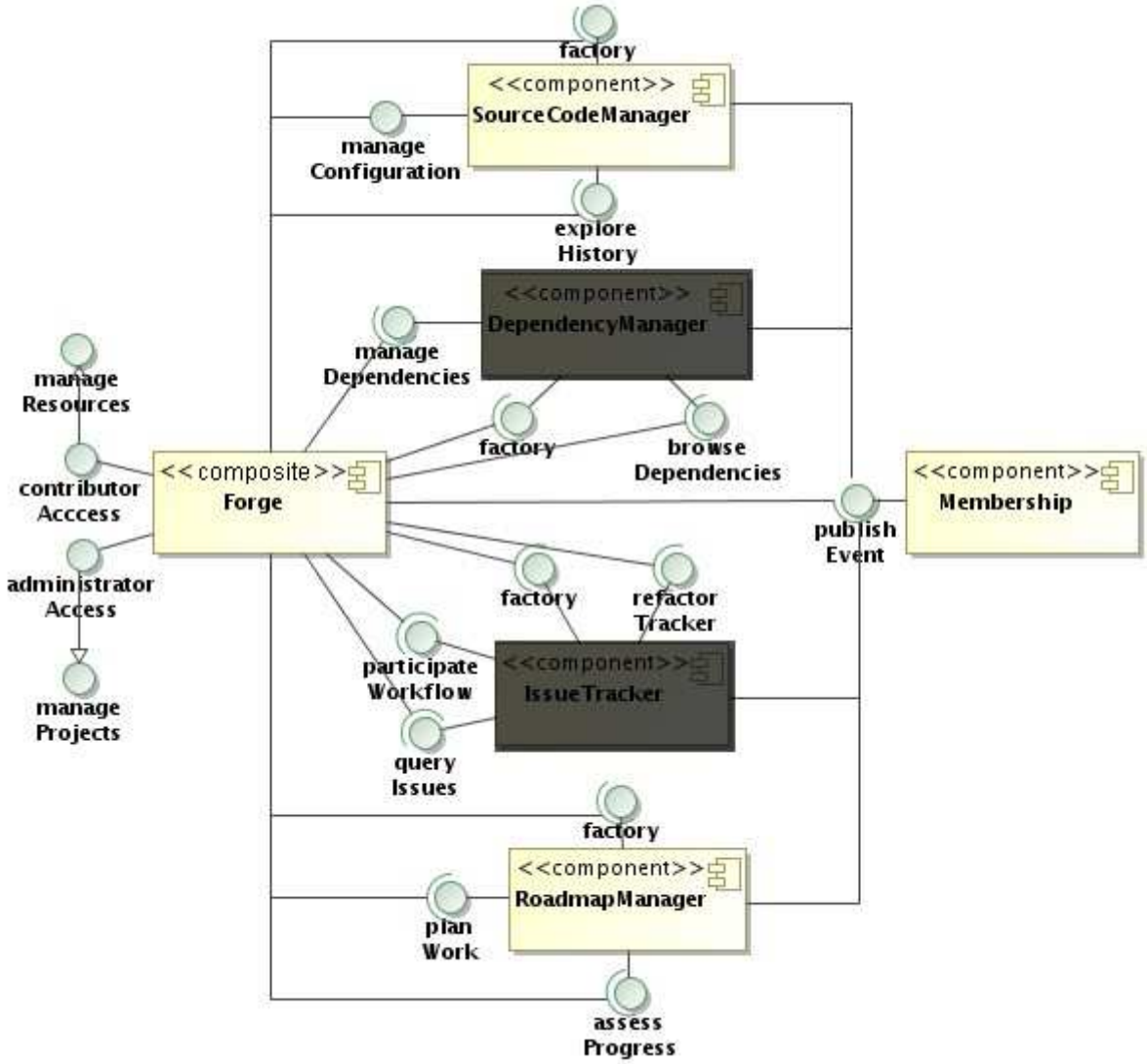


Illustration 20: Collaborative development: forge components (implementation view)

4.3.2 Quality check scenario realization

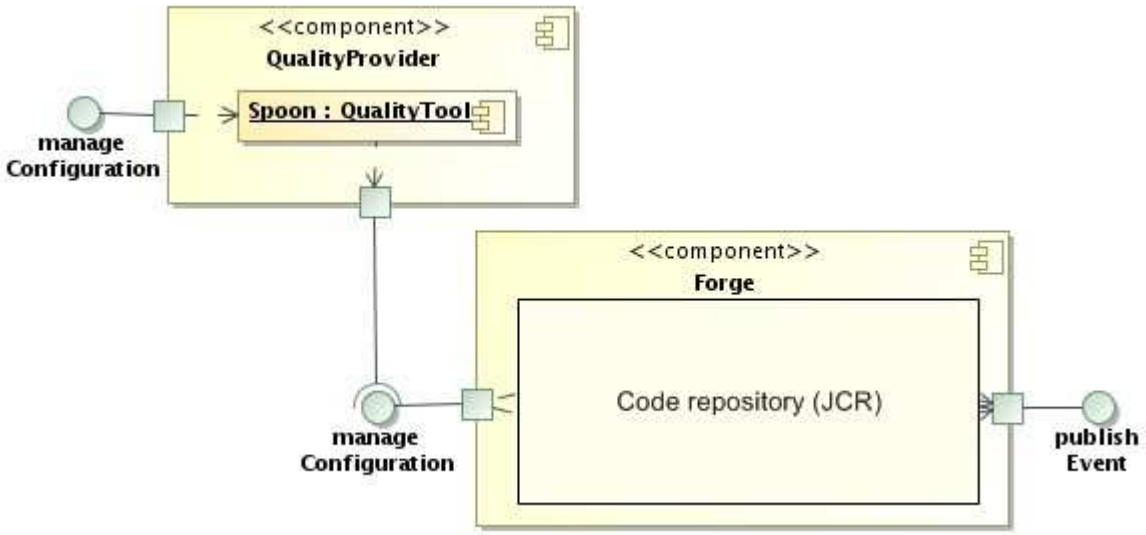


Illustration 21: Collaborative development: forge scenario #1 (quality check)

4.3.3 Development project progress dashboard scenario realization

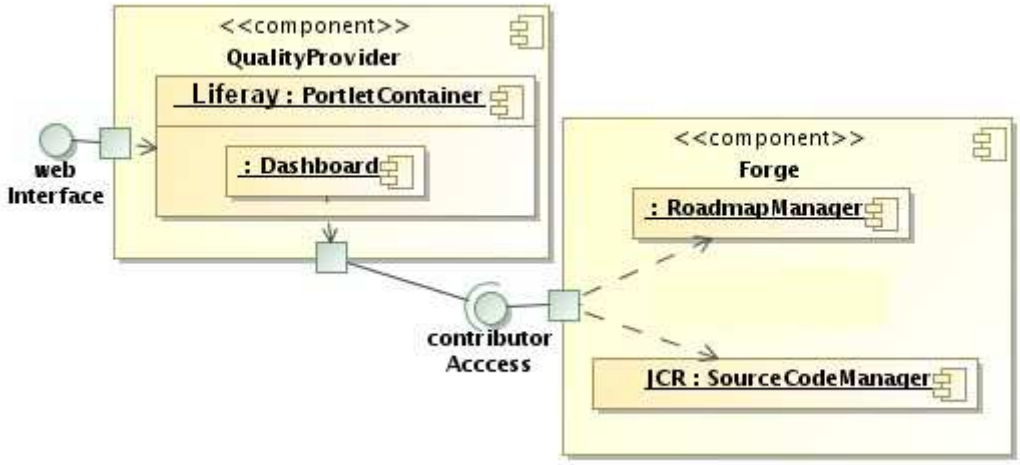


Illustration 22: Collaborative development: forge scenario #2 (project progress dashboard)

4.3.4 Project portal components

In addition to the forge components, the project portal solution features:

- a CMS component
- built on top of an ECM component (which also provides the JCR standard ECM API), which manages use cases, features and information about project integration. Note that it is linked to the source code versioning system within the forge, since source code and releases of project integrations are also hosted by it
- a voting system, built on top of a workflow component.

4.3.5 Forge: SCA design

4.3.5.1 Global design

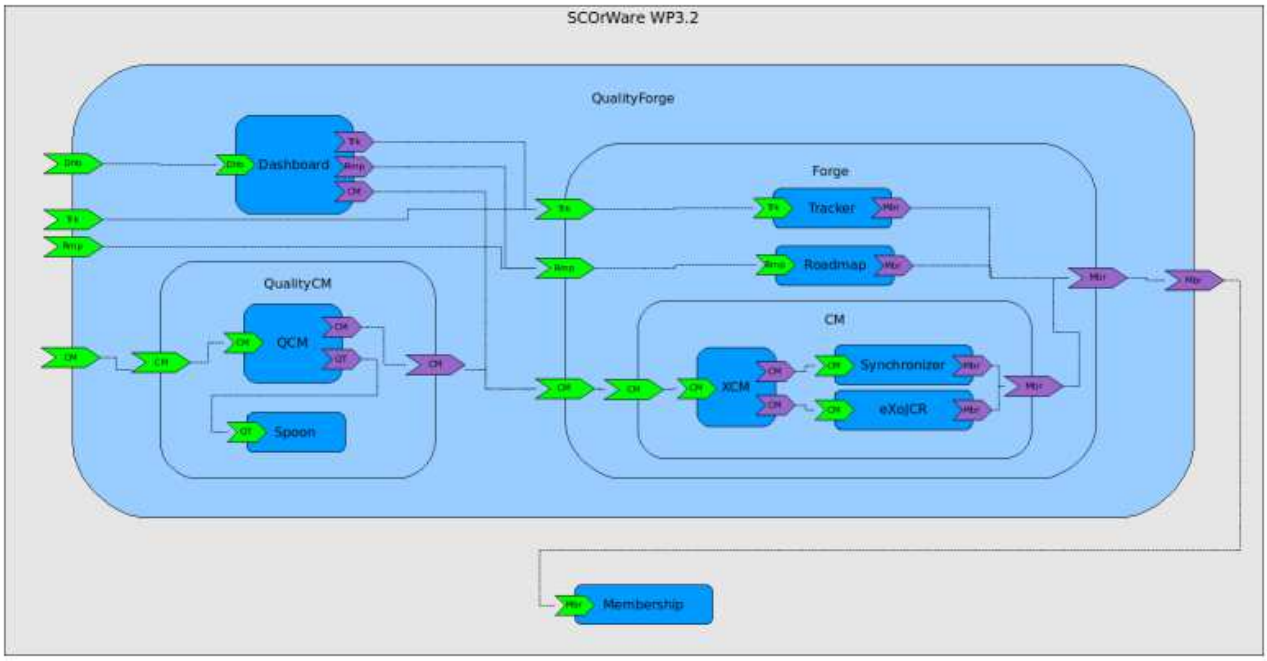


Illustration 23: Collaborative development: SCA view of a quality forge (global design)

This SCA global design is consistent with the design global view, as shown in Illustration 19.

4.3.5.2 Implementation design

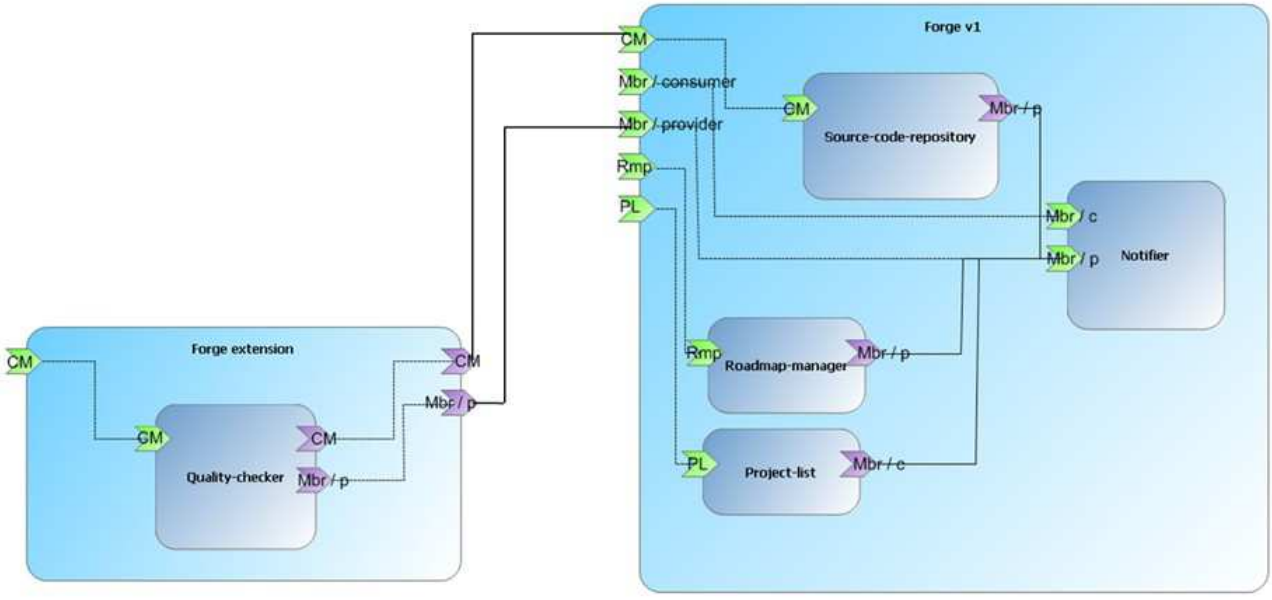


Illustration 24: Collaborative development: SCA view of a quality forge (implementation design)

This SCA implementation design is consistent with the design implementation view, as shown in Illustration 20.

4.3.5.3 Services

- **Rmp** (Roadmap): project planner browse and query interface
- **CM** (Configuration Management): configuration management (providing or not quality check) and content access interface
- **Mbr** (Membership): event publishing interface
- **PL** (Project List): existing projects with name and description

4.3.5.4 Components

- **Notifier**: provides features for team building and awareness. These features are split into 2 services:
  - *producer*: provides all the functionalities for sending messages
  - *consumer:* provides all the functionalities to subscribe to a kind of message
- **Source-code-repository**: is able to store source code from different projects and support file versioning
- **Roadmap-manager**: is able to define different milestones themselves defined by a list of tasks.
- **Quality-checker:** allows to check if the committed code is well commented
- **Project-List**: manages the list of existing projects

4.3.5.5 Composites

- **Forge v1**: includes all the functionalities of a "basic" forge
- **Forge extensions**: this composite extends the functionalities of the "Forge v1", by adding "quality-check", for checking if the source code contains some comment

4.3.6 Project portal: SCA design

4.3.6.1 Global design

Hereunder is the initial and global design, provided as a SCA diagram for the Project Portal solution.

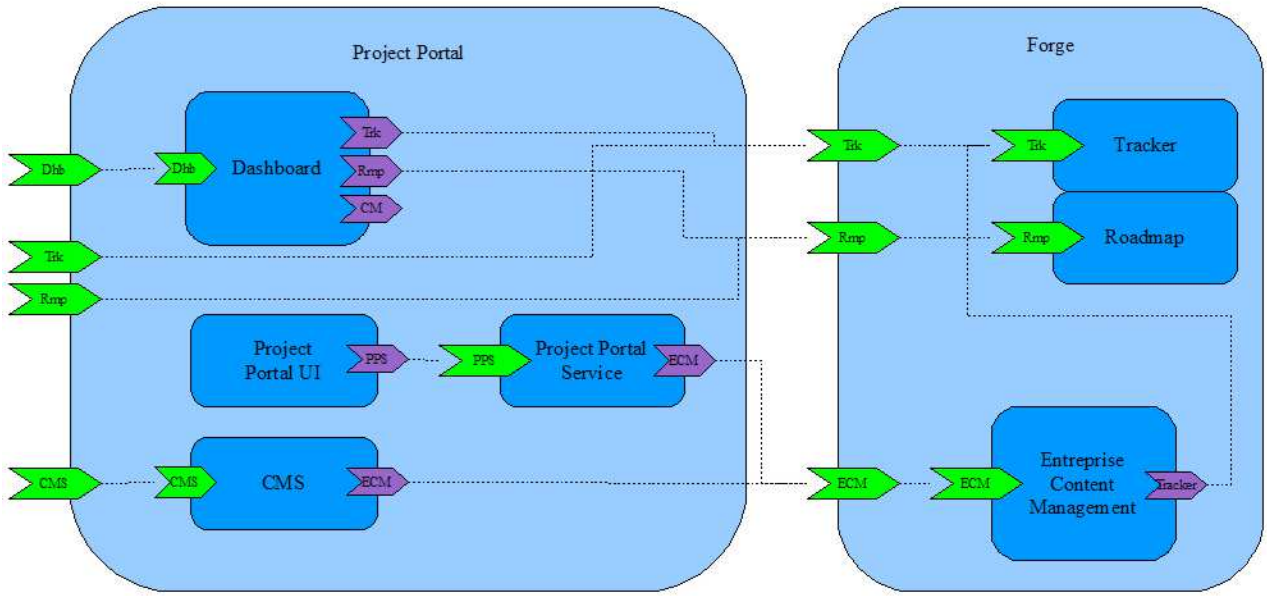


Illustration 25: Collaborative development: SCA view of a project portal (global view)

Finally, because of some lack of time and resources for implementation, the UI and Portal components have not been designed and developed as SCA ones, but are reusing services as exposed by the SCA Forge, thanks to the SCA binding mechanism. And so, the Dashboard component, as required by the scenario as shown in Illustration 22, is a set of portlets, which are external to the SCA domain in our case.

The implementation design view is shown below.

4.3.6.2 Implementation design

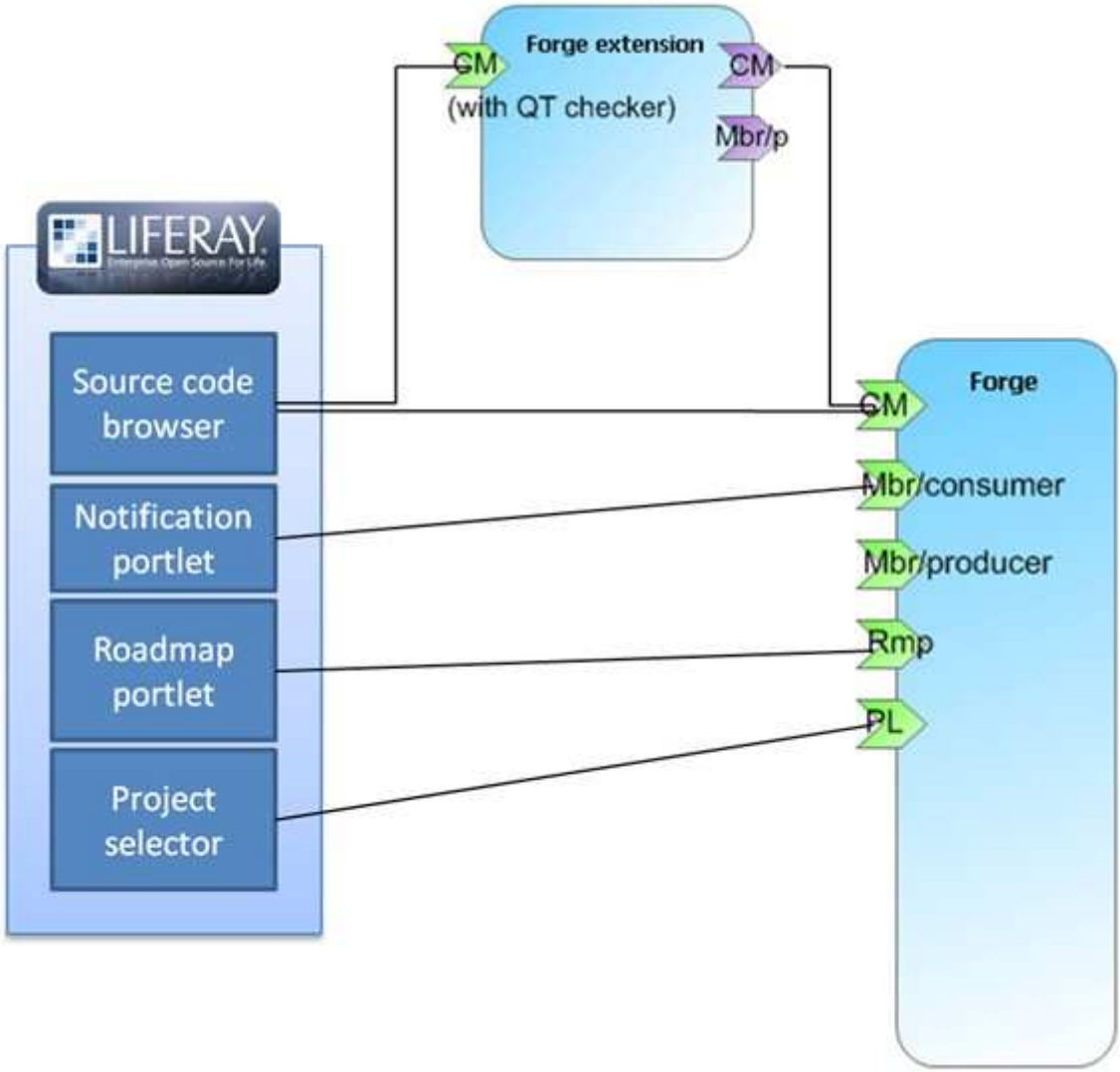


Illustration 26: Collaborative development: SCA view of a project portal (implementation view)

4.3.6.3 Components

The final implementation is available and documented on the OW2 Scarbo website at http://scarbo.ow2.org, as an official Scarbo demonstrator. It has required the demonstrator to be migrated from SCOrWare to the OW2 consortium.

- **Project Portal UI** : UI within the Project Portal for Project Portal Services.
  - The Portal standard in Java is the Portlet specification. It is implemented by various Portal components that are available in Open Source, like Liferay, jBoss Portal and eXo Portal. Liferay is the most well-known and feature-rich one, moreover available under the liberal MIT license. For these reasons in addition to its own expertise, Open Wide Liferay as the portal of choice for OS2P.
  - On the UI level (which is one of the main areas adressed by Portlets and Portals), the OS2P UI is developed as portlets using the Liferay SDK. We choose Struts as the UI technology of choice for presentation parts that are native to Liferay, the reason being it is still the most used UI java technology and not much complicated which allows to focus on

more service-oriented topics.

- On the level of portlet definition (the other key area of the Portlet domain), the standard way to define portlets, extended by Liferay's specific features, are used for OS2P. This has first been adressed by Open Wide in standard Portlet fashion, by developing Portlets that are able to use a flexible service layer as defined below. Once specified, first the whole OS2P architecture has been developed on Tuscany, then on FraSCAti (Java source-based) prototype, and finally on FraSCAti final 0.5 version (which includes the famed generation factory to avoid the requirement of providing java sources of classes mentioned in composites).
- In addition, on an idea of INRIA Tuvalu, and with mentioned interest and support of INRIA Adam, Open Wide specifies and implements an SCAProxyPortlet allowing thanks to the SCA standard and its FraSCAti implementation to define the (Portlet interface-implementing) class of a portlet, specify its properties and inject services in it. It is done by developing an SCAProxyPortlet class implementing Portlet using the delegate pattern, and loading a well-defined composite and calling a known Portlet-returning service within it in order to set this delegate. Additional customization can be achieved by subclassing the SCAProxyPortlet. In order to use it, the user must develop its portlet as usually, but set it up in the right SCA composite and configure the portal to rather load SCAProxyPortlet It is demonstrated in the implementation of some OS2P use cases. NB. This replaces other ideas that have been studied but not developed because less interesting.

- **Project Portal Service (PPS)** : business service for project portal domain, providing use cases, requested features and project integrations management and linking
  - The service layer is developed as a simple service-oriented, java interface and bean implementation-based service-oriented application architecture. This architecture is assembled by applying service injectionpatterns, both for local services (other services of the PPS business layer) and for remote (ex. Workflow engine, forge, ECM) services. Open Wide specifies and implements in OS2P various ways of doing both, be it using Spring (the most widely used IoC container), SCA (which can be seen locally as IoC itself), mere Java code, or even the Spring SCA implementation for FraSCAti as developed by the INRIA.
  - The conclusion is that SCA (with its powerful binding definition) is best used for services defined in the whole information service's SOA, while Spring (and its numerous, toolbox-like helpers) is appropriate to assemble services that are local to the application architecture. However, using the above-specified SCAProxyPortlet to define a portlet's implementation and injected SOA services, coupled to Spring applicative services injected in SCA through the SCA Spring implementation brings the benefits of both worlds.
- **Workflow engine and tooling** : this is required for all approval and more widely workflow features of OS2P.
  - For this, Eclipse JWT (Java Workflow Tooling) is used on the tooling side, while OW2 Bonita is used on the runtime (workflow engine) side. Bonita has seen a major evolution lately, going from Bonita 3 to Nova Bonita (4) ; Open Wide has gone along with it, first integrating Bonita 3 in year 1 and then Nova Bonita in year 2 in OS2P.
  - This includes providing through SCA a local version of the Bonita-implemented JWT WorkflowService, in order to allow OS2P to start and interact with workflows. The WorkflowService has first been implemented using EJB since Bonita 3 was based on such an architecture, while the switch to an implementation local to Bonita has been preferred by Open Wide in the new Nova Bonita since its new lightweight and embeddable architecture allows it.
  - It is to be noted that the new embeddable architecture also allows it to be deployed two ways : either in platform mode within a webapp with its dedicated web management

console (out of the box), or embedded in an SCA FraSCAti-deployed simple web application (where FraSCAti is deployed by either an SCAServlet on an SCAContextListener), which Open Wide develops and provides.

- **CMS : CMS features built on top of the ECM container, see below**
- **Enterprise Content Management (ECM)** : ECM component.
  - Various ECM components are available in Open Source, like Alfresco, Nuxeo and eXo ECM. Alfresco is the most well-known, comprehensive, rich one. It is available under the GPL license, has a service-oriented application architecture and implements standards such as WebServices, JCR and the upcoming CMIS. For these reasons in addition to its own expertise, Open Wide choses and integrates Alfresco as the ECM of choice for OS2P.
  - To achieve this portal-ECM integration in a service-oriented way, various technologies are available. On the business domain side, JCR is rejected because of complexity, lack of market penetration and possible obsolezcence. Because of new standards like CMIS (Content Management Interoperability Standard). WebServices are well supported by SCA (including FraSCAti) and most technology stacks, while REST-based protocols offer a lighter, more agile alternative. In order to implement said use cases while studying various cases of service integration within a portal, Open Wide uses WS for deep, service-level integration and REST for light, UI-level integration.
  - Integrating Alfresco through a business or technical standard is especially interesting. For this the most meaningful are WebServices on the technical side and CMIS on the business side. To achieve it, possible architectures include SCA bindings but also SCA's appropriateness for assembling services of a given layer of granularity or business level to provide services of another such layer. On the basis of these strengthes Open Wide specifies and prototypes such an integration of FraSCAti-based SCA and Alfresco.

4.3.6.4 Composites

- Project Portal (including Use case and success story portlet) : built as a Struts-based SCAPortlet that loads
- Project content portlet : configured using SCAProxyPortlet, built using light ECM integration
- Various versions of the previous ones providing various alternative configurations, and other "helper" composites

4.3.6.5 Project development

- Liferay, Alfresco and even Bonita are rather development platforms than mere components. This means that they have their own (ant-based) build mechanism and development practices, which means integrating them together and with such a wide range of middleware as FraSCAti and Tuscany (whose build are maven-based) are themselves integrating quite a challenge. Open Wide has studied this challenge in year one, and answered by a complete build mechanism integrating maven for the SCA part and ant for the Portal and ECM parts, relying on the maven-ant integration and making heavy use of "excludes" to work around dependency conflicts between all integrated packages. This has first being done in first year on Tuscany, then on FraSCAti's prototype, and finally on FraSCAti final 0.5 version.
- Integration with the Forge perimeter is done in standard Portlet fashion. Forge portlets (Forge notification and Forge repository command) are made available right along OS2P portlets in the portal.

4.4 SCA specification coverage

Assembly model	***Simple component***	***Service***	*Rmp, CM, QT, Mbr, PL, OS2P*
		Reference	*Rmp, CM, QT, Mbr, PL*
	Composite	***Service***	*Rmp, CM, Mbr, PL, OS2P*
		Reference	*CM, OS2P*
		Component	*Notifier,Source-code Repository, Roadmap-manager , Quality-checker, Project-List, OS2P*
		Composite include	*<none>*
Java Component Implementation			*Notifier, Source-code Repository, Roadmap-manager , Quality-checker, Project-List, OS2P*
Java Annotations and APIs			*Used for implementation of the following services: Rmp, CM, QT, Mbr, PL, OS2P*
Web Services Binding			*Rmp, CM, QT, Mbr, PL, OS2P*

Table 3: SCA coverage (collaborative development)

Component	X
Composite	X
Wire	X
Property	X
Binding Web Service	X
Service (Local & Remotable)	X
Domain	-
Interface	X
Implementation (Java)	X

Table 4: SCA assembly model specification (collaborative development)

@Service	X
@Reference	X
@Property	X
@Scope	X
@Remotable	X
@Conversational	X
@OneWay	X
@AllowsPassByReference	-
@Callback	X

@ComponentName	-
@Conversation	-
@Constructor	-
@Context	-
@Destroy	-
@EagerInit	-
@EndConversation	-
@Init	X
@ConversationAttribute	-
@ConversationID	-

Table 5: SCA Java common annotations and APIs (collaborative development)

4.5 Lessons learned

4.5.1 FraSCAti vs. Tuscany

- Binding as provided by FraSCAti allows a fine-grained configuration for Web Services, unlike Tuscany
- Memory footprint: our tests done on a development version of the Forge have given a smallest one for FraSCAti vs. Tuscany (92 Mo vs. 101 Mo)
- Error messages: for this topic, some enhancements should be done in FraSCAti, as error messages appearing in Tuscany are more detailed and clearer
- @Remotable: this annotation is not mandatory in FraSCAti for exposing a service with a binding, and so it facilitates a standard interface, which could keep unchanged

4.5.2 Benefits of SCA

Benefits of SCA are the following ones, according to the two stages where SCA is used.

4.5.2.1 At design stage

- standardized architecture approach
- conceptual standardization
- normalization in the graphical representation mode
- component (code base) reusing
- concrete service reusing.

4.5.2.2 At development stage

- the annotations allow to decrease the number of coding lines and offer high-quality services
- binding mechanisms allow to expose and use services with different communication protocols, without any extra cost in programing
- there is no impact of the protocol as chosen on the component itself .

5 REUSE AND ENRICHMENT OF COMPONENTS CORPORATE FRONT-END LINKED WITH SAP BACK-OFFICE (Task 3.3)

Change history vs. version 1.0: the first version 1.0 (October, 2007) of the specification was describing the first implementation of this usage demonstrator, using the Tuscany SCA platform as deployment and runtime platform. This final version specifies the final usage demonstrator, enriching it with the extensions as required following the first year project review with ANR (and its expert), including extended functionalities (e.g. a SAP real platform), demonstration of the capabilities of the SCOrWare platform both in terms of functionalities, and volumetric, deployment on the SCA platform (both FraSCAti and PetALS). Following the first year project review, and as required by the ANR and its expert, this usage demonstrator is considered as the main one ("Pilot") for a complete demonstration of the SCOrWare platform.

5.1 Introduction – Context of the Business

EdifiXio implements custom applications for some large industrial groups. Most of these applications are designed and specified in collaboration with Corporate and then deployed and adjusted for each Business Unit of the group. Of course, the Business context is very different for each Unit and implies a high capacity of modulation and adjustment.
EdifiXio has developed a specific methodology to address this specific business and improves it all the time in order to be always more efficient and competitive on this sector.
The key factor is to continuously decrease the cost of the customization and the deployment of a new instance of the application. The cost of the maintenance enhancements is also on the critical way.

A major technical step has been made in the last years with the ability to run all the Business Unit instances of the application on a single runtime with a single version of source code. This improvement has been lead by the JEE standard.
The current step of improvement is based on the modularity, the assemblage of parametrized business modules in order to achieve a high level of re usability of the existing business code. The JEE standard is our main technical support for doing so. Unfortunately, JEE does not provide all the expected flexibility.

The task 3.3 of the SCOrWare project gives us the opportunity to study how the new SCA standard and more specially the SCOrWare platform, could provide us with more flexibility and more efficiency. The contribution of EdifiXio will focus on the two following topics :

- Reuse and enrichment of components.
- SCA and JEE.

Both will be covered in a demonstrator specified in this current document.

In this document, we assume that the following technical points are managed in the SCOrWare platform even if we still don't know how they will be implemented and if they will be part of a demonstrator. These points are mandatory for our business:

- Transaction Management and especially the Two Phases Commit.
- Automate integration chain and especially a first level of quality assurance including unit testing execution and code coverage estimation for business applications.
- Several target environments are operational (Tuscany/Tinfi/FraSCAti Standalone/Integrated Tomcat/JOnAS, …)

5.2 Functional Specification

5.2.1 Introduction

Our demonstrator will be a Corporate Front-End Order Management application linked to Back

Offices.

Architecture should be adapted to Production Application even if the implementation is limited to the Demonstrator scope.

A production application is composed of a single Business Logic connected to several Back Offices. This application is accessible from Internet through different ways : PDA, Portal, Standalone WebApplication, Web2 Application.
Here is a presentation of a standard architecture on production:

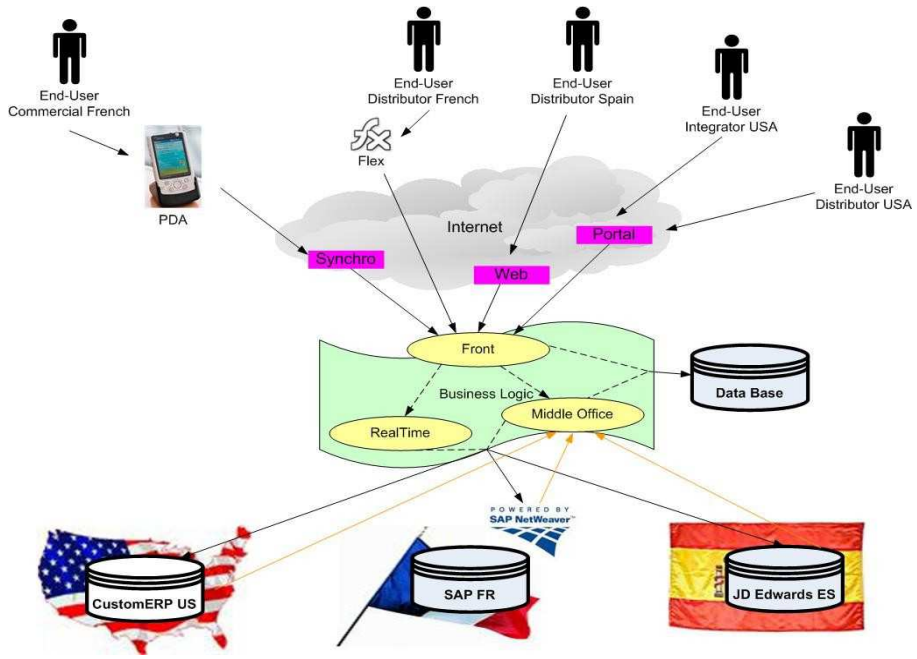


Illustration 27: production application presentation

For demonstrator purpose we will

- Limit the application accessibility to the standalone Web Application and the Web2 Application.
- Use only simulated Back Offices. One of them (French one) running on a SAP Netweaver Application server.
- Use a single user audience on 3 different Business Units (France, USA and Spain).

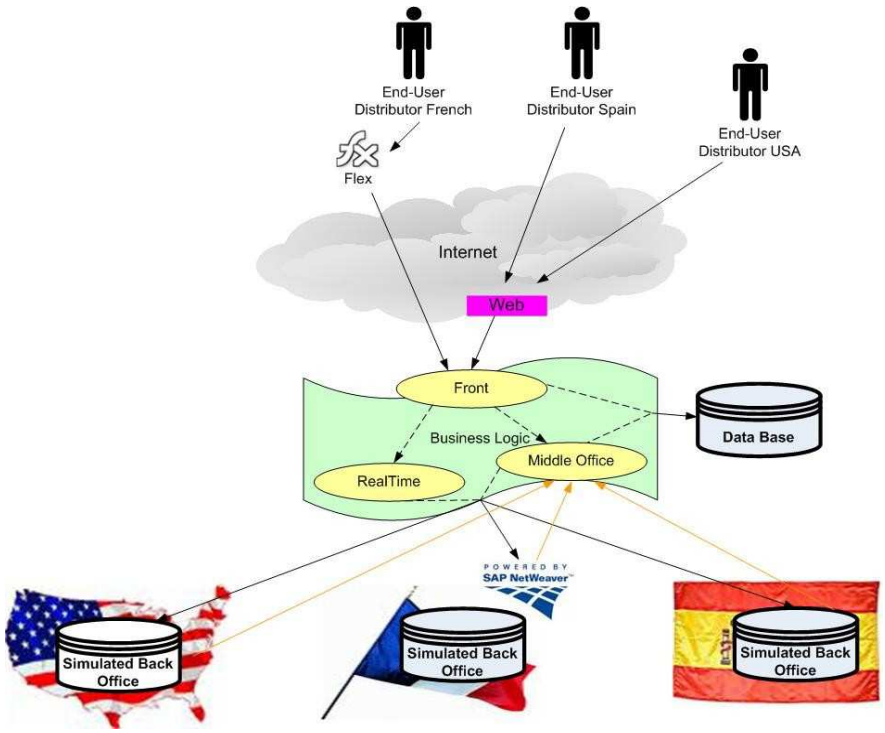


Illustration 28: demonstrator application presentation

As a result, the demonstrator will be composed of a Web Application and Web2 Application linked to a set of Front Order Management services (Delivery Notes, Price and Availability, Order Status and Order Entry) using Local Database and Back Offices. Part on this set of Front Order Management services (Delivery Notes) will be kept in a JEE architecture. Back Office will be called through Real Time flow (Price and Availability) and asynchronous flow through Middle Office (Order Entry). Moreover, the Back Offices will be scheduled to send data to populate Front Local Database through the Middle Office component (Order, Product and Price).

User and Account management (provisioning, right, security …) will not be part of this demonstrator. A set of User and Account will be previously created.

5.2.2 Web Application

The Web Application will be composed of these following screens

- Delivery Notes
  - Delivery Notes search screen

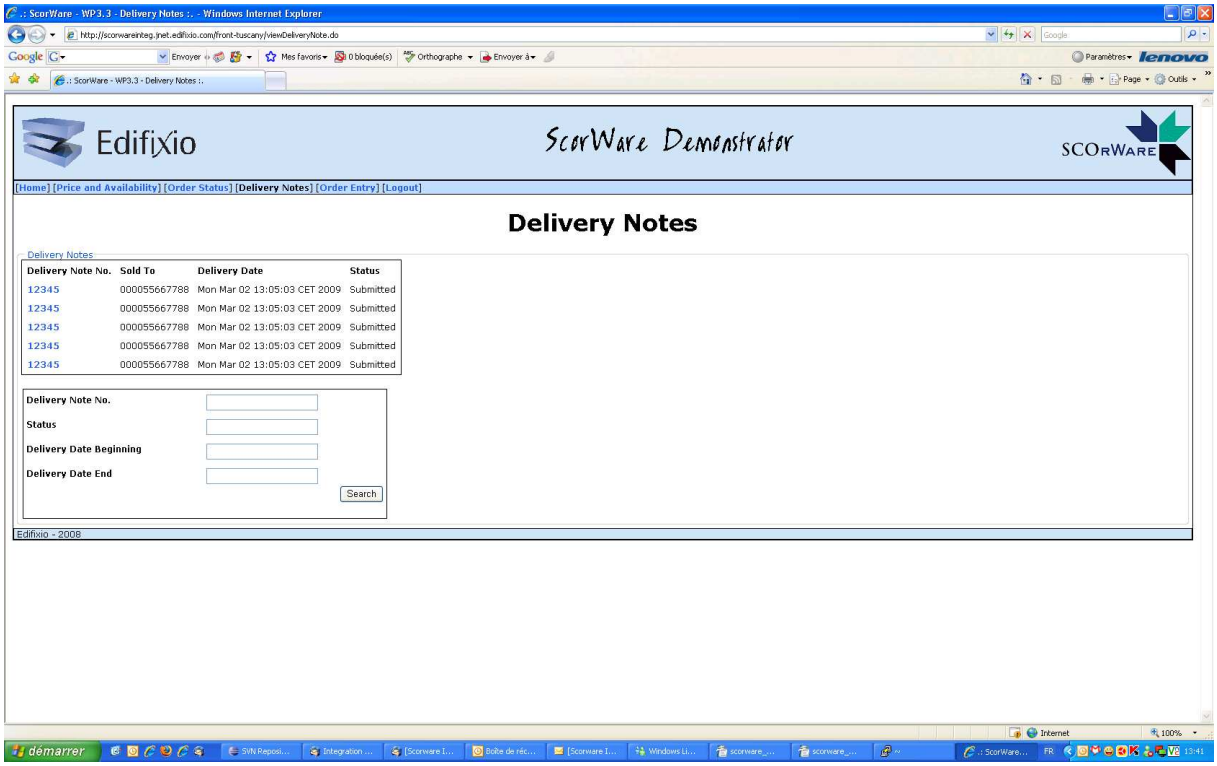


Illustration 29: delivery notes search

- Delivery Note detail screen

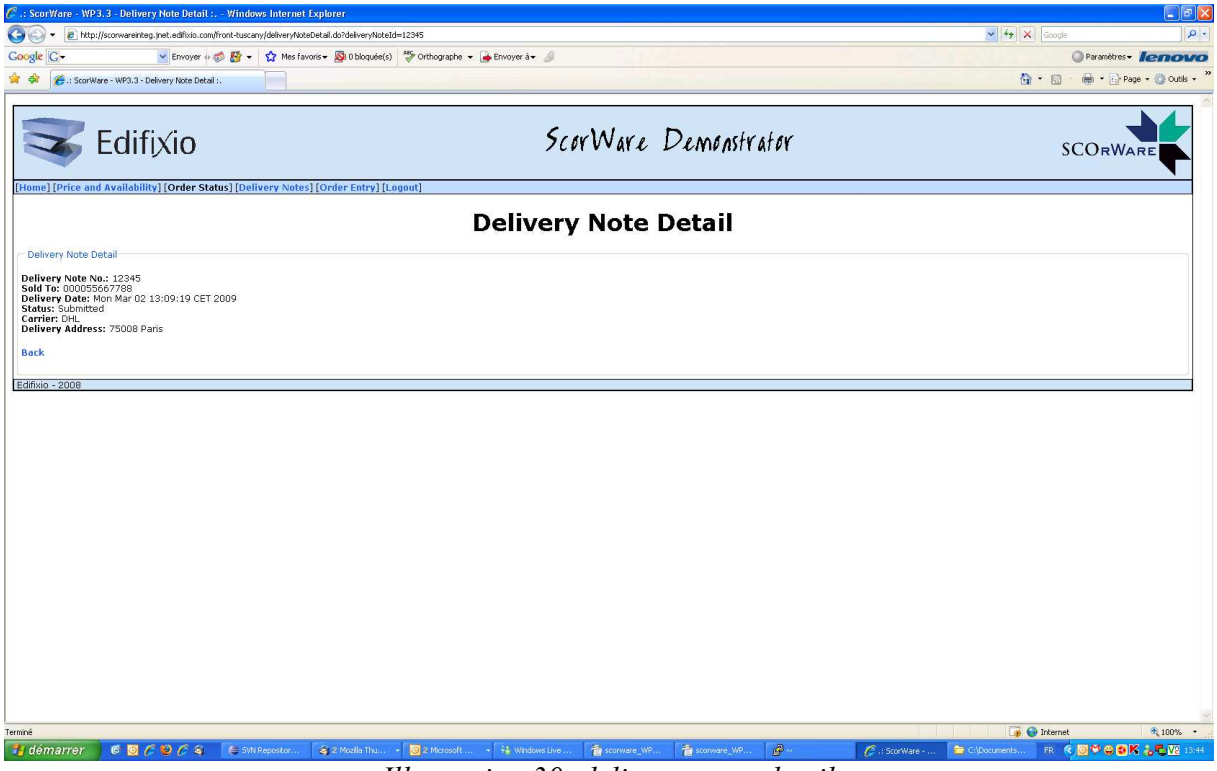


Illustration 30: delivery notes detail

- Price and Availability
  - Price and Availability request screen

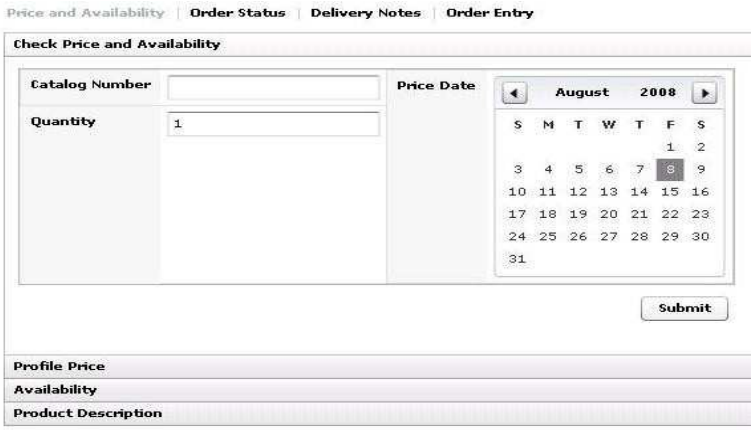


Illustration 31: price and availability request screen (Web2 Application)

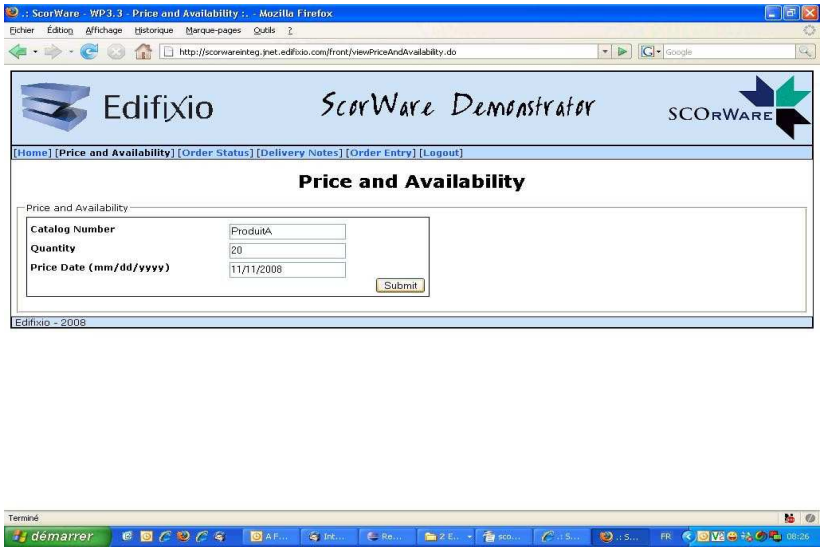


Illustration 32: price and availability request screen (Web Application)

- Price and Availability result screen

Illustration 33: price and availability result screen (Web2 Application)

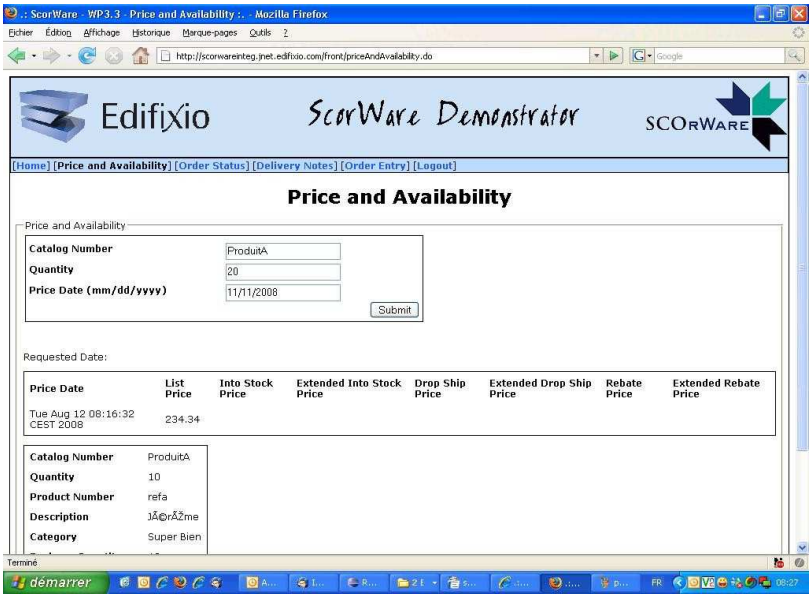


Illustration 34: price and availability result screen (Web Application)

- Order Status
  - Order Status search screen

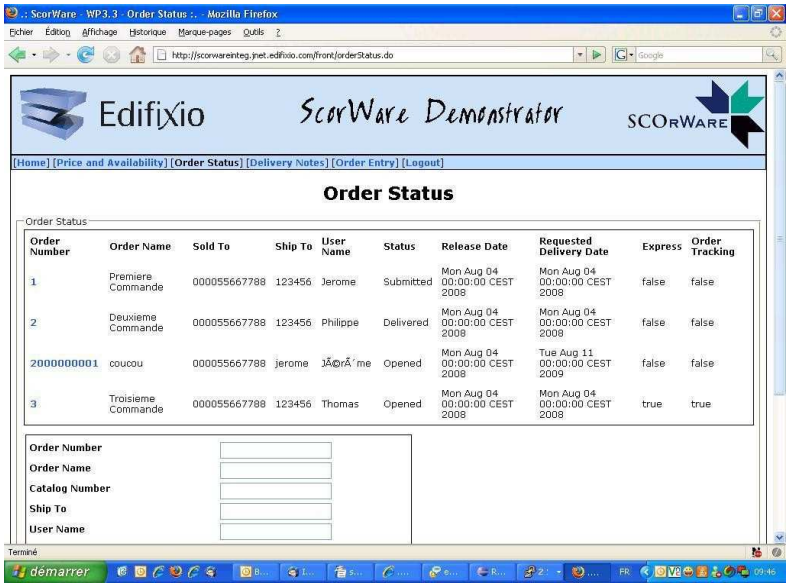


Illustration 35: order status list screen (Web Application)

- Order Status detail screen

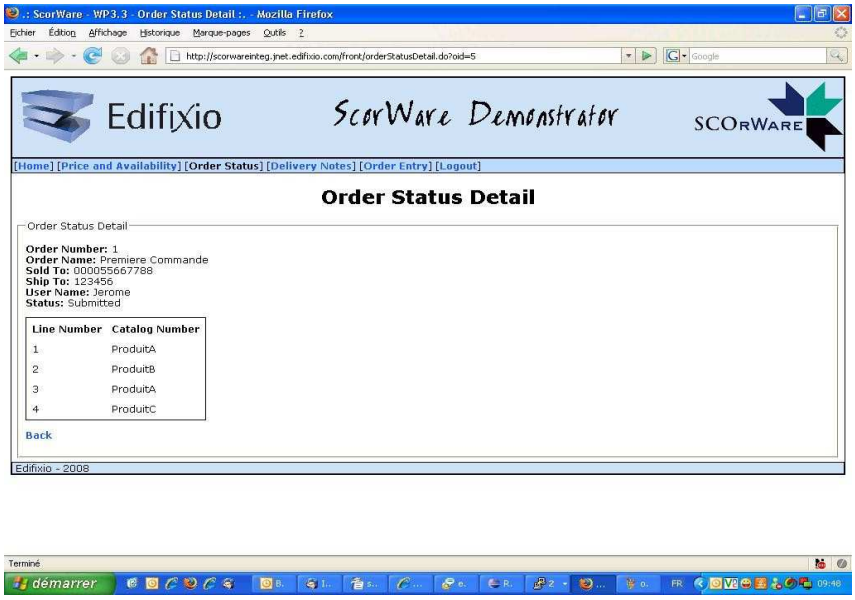


Illustration 36: order status detail screen (Web Application)

- Order Entry
  - Order Entry screen

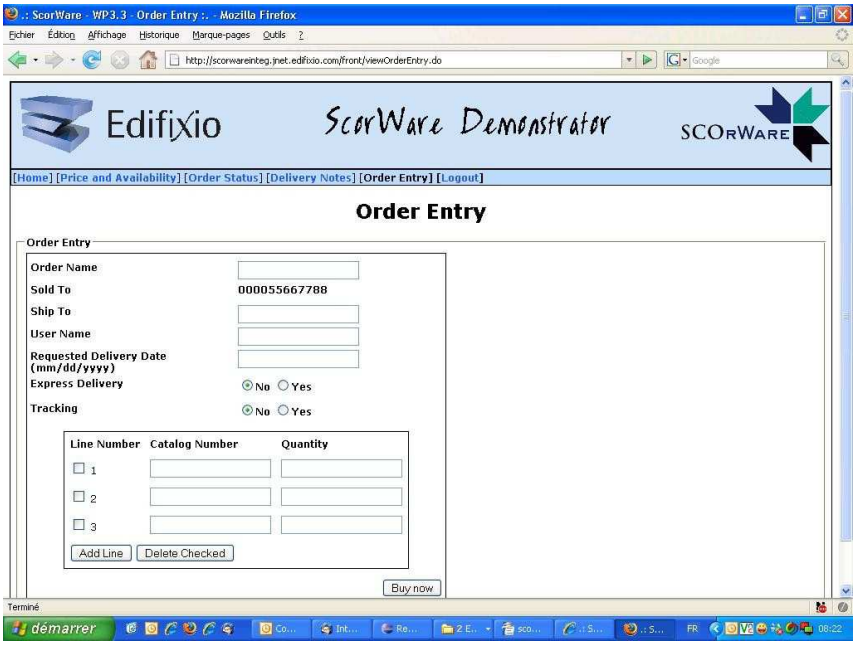


Illustration 37: order entry screen (Web Application)

5.2.3 Front End services

The Front End will be divided in 4 services: Delivery Notes, Price and Availability, Order Status and Order Entry.

5.2.3.1 Delivery Notes

This service provides the Delivery Notes on a given Account. This service is composed of 2 functions:

- the last n Delivery Notes of an Account. The Delivery Notes information is coming from Front Local Database.
- the full detail of a Delivery Note of an Account. The Delivery Note information is coming from Back Office through the Real Time service.

5.2.3.2 Price and Availability service

This service provides the Price and Availability of a given Product for a given quantity and a given date.
First the Service checks if the Product exists in the local database then it tries to retrieve the full price and availability information of this product through the Real Time service. If Real Time failed (Back Office unavailable for instance) then full Product and Price information are retrieved from Front Local Database (rescue mode).

5.2.3.3 Order Status service

This service provides the Order Status on a given Account. This service is composed of 2 functions:

- the last n Orders of an Account. The Orders information is coming from Front Local Database.
- the full detail of an Order of an Account. The Order information is coming from Front Local Database.

5.2.3.4 Order Entry service

This service will make an Order Entry on a given Account for a given User. It allows to fill a Shopping Cart (Header and Lines) and to purchase this Shopping Cart.
The Purchase functionality will first store the Shopping Cart (which became an Order) into Front

Local Database and then send it to the Middle Office service. Then, the end-user can follow his navigation while in an asynchronous mode, the Middle Office service sends the Order to the Back Office.

Note: storing Shopping Cart into Front Local Database and then sending it to the Middle Office service should be in the same transaction. As we are using two persistence systems (Middle Office service one and Front Local Database one), the transaction should be inside a two phases commit mechanism. If one failed, both are roll backed.

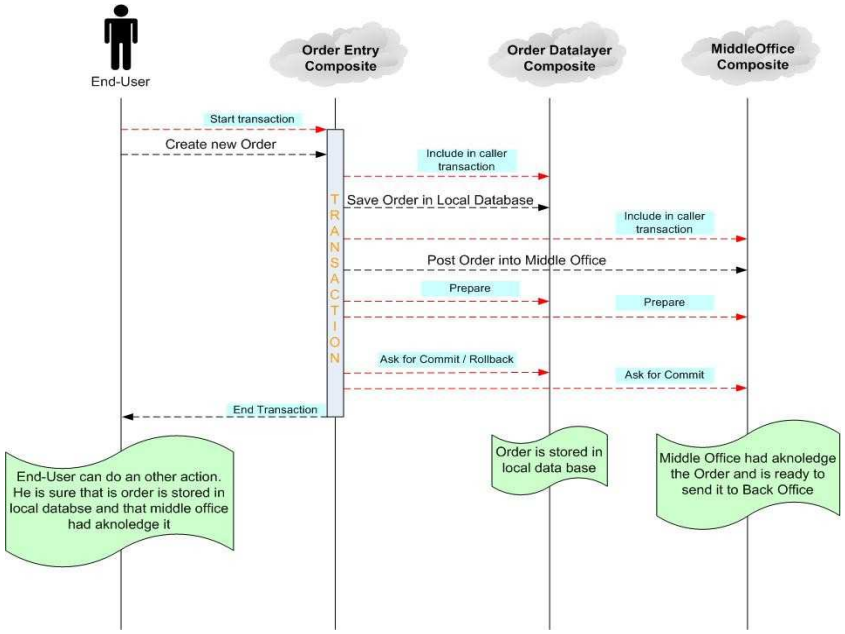


Illustration 38: order entry creates order 2-phase commit transaction

5.2.4 Real Time service

This service will perform a Real Time call to Back Offices. It's just an intermediary between Front Services and Back Offices.

5.2.5 Middle Office service

This service will be an asynchronous Middle Office between Front Services and Back Offices (Order Entry) or between Back Offices and Front Office through the Front Local Database (Order, Product and Price).

5.2.6 Front Local Database

This database will store the Order, Product, Price, User and Account information. User and Account will be initialized in the Demonstrator installation procedure. Product and Price information will be filled with the Back Offices information. Order will be filled with Front and Back Offices information.

5.2.7 Back Offices

In this demonstrator, we will use three simulated Back Offices (Business Unit France, Spain and USA). One of these simulated Back Office will be working inside a SAP Netweaver Application server. These simulated Back Offices will have the same interfaces as a real one (SAP for instance).

5.3 Methodology

During the implementation of the demonstrator, we propose to apply as much as possible our best

practices and to report any practice that has to be changed.

5.3.1 Target Environments

The demonstrator and more generally a real project may be developed by several developers in several separated teams. As a result we need to be able to provide an adapted working environment for development purposes and integration between the different teams. We also have to use a qualification environment closest as possible to the target production environment. The Pre-Production environment allows us to validate the hosting infrastructure with an already qualified code version (qualified in qualification environment). Furthermore, it allows the Operation Team to train for the Production installation.
Moreover, for the demonstration purpose, a demo environment has to be set.

Environment\Quality	Performance	Hardware Cost	Hosting Cost	Installation Time
Demonstration	medium	cheap	cheap	short
Development	medium	cheap	cheap	short
Integration	medium	medium	cheap	medium
Qualification	good	expensive	cheap	long
Pre-Production	medium	medium	expensive	medium
Production	very good	very expensive	expensive	long

Note: for the demonstrator, we won't have qualification, pre-production and production environments. However, we describe them in order to illustrate the constraints of environments we may have in a real project.
Check points:

- ✓ Only Environment Parameters need to be changed during the transportation from one environment to another.
- ✓ Environment Parameters are well separated from Architecture Parameters and Applicative Parameters.

5.3.1.1 Demonstration environment

The aim of this environment is to allow the commercial to do a demonstration of the demonstrator in a simple environment (no internet connection …).
As a result we will use a Windows laptop where all components are installed inside. The Back offices are Simulated ones.

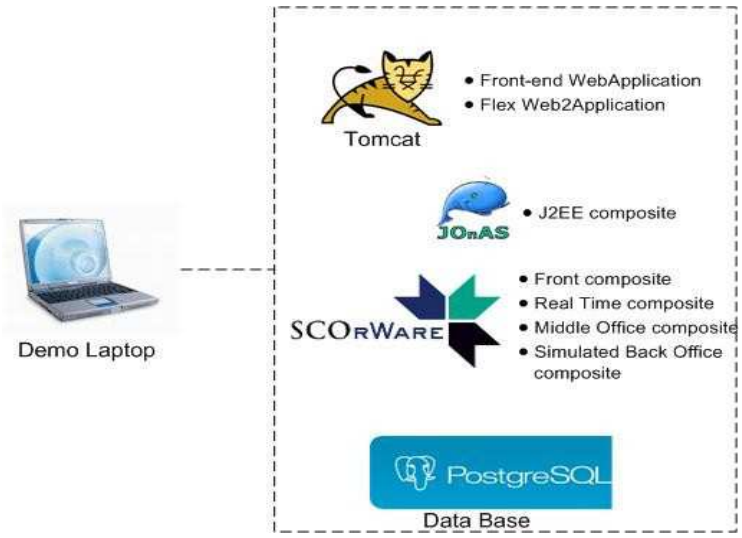


Illustration 39: demonstration environment

5.3.1.2 Development environment

The aim of this environment is to allow several front developers to work in parallel in this demonstrator. Consequently, each developer needs to have its own environment. Only database will be shared between developers.
As a result, we will use a Windows computer where all components are installed inside, except the database. The Back offices are Simulated ones.

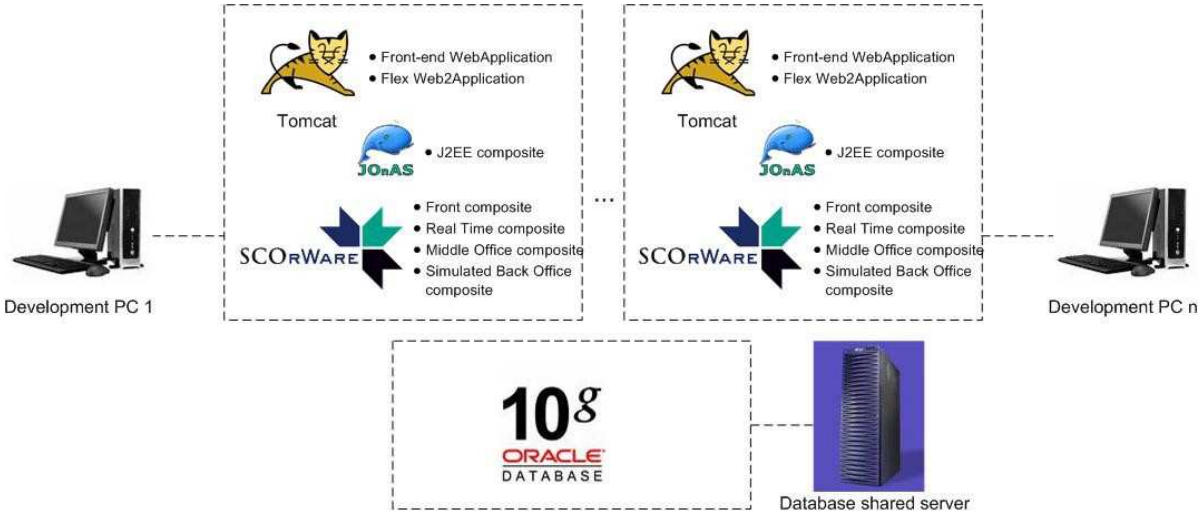


Illustration 40: development environment

5.3.1.3 Integration environment

The aim of this environment is to allow the integration between Back Offices teams and front developers team(s).
We would use a Linux server for front purpose. This server would be connected to its database and to Back Offices servers. For demonstrator purpose we won't use real Back Offices but we will deploy a simulated Back Office in a Sap NetWeaver Application server

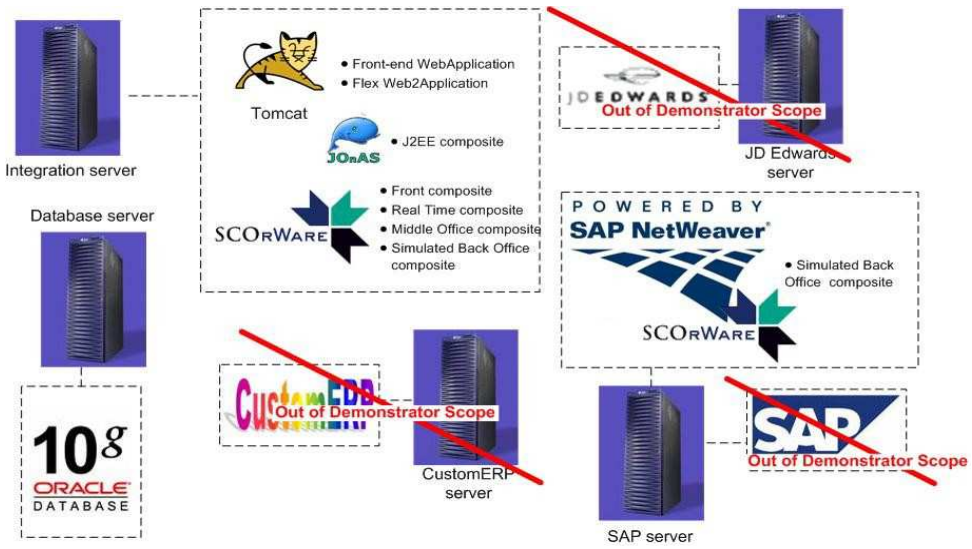


Illustration 41: integration environment

5.3.1.4 Qualification environment (out of demonstrator scope)

The aim of this environment is to allow Operation and Qualification teams to install and test the Application. The Qualification environment has to be as close as possible to the target production environment. This environment won't be used for the Demonstrator.
We would use several Unix servers for front purpose. These servers would be connected to the database and to Back Offices servers.
All servers are duplicated for fail over and load balancing mechanism.

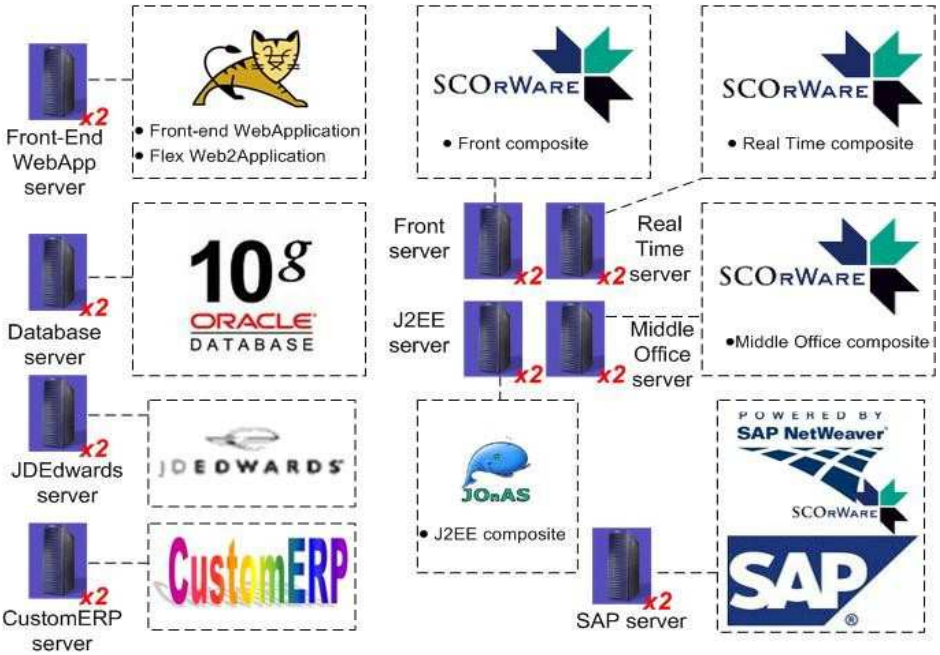


Illustration 42: qualification environment (cluster) – out of demonstrator scope

5.3.2 Tools

Here is the list of tools we want to integrate with the Project life cycle.
The developers will use in order to

- develop Eclipse with SCOrWare tool plugins
- compile Maven

- validate unitary his work Junit
- archive SVN

Thanks to shell scripts (or better tools), the integration environment will be able to

- install all the desired software:
  - FraSCAti, Tuscany, Tomcat, JOnAS
  - Maven for compilation
  - Emma for coverage
  - Junitee for integration test
- build a new version coming from SVN thanks to Maven and provide a full report on code quality thanks to Emma and Junitee

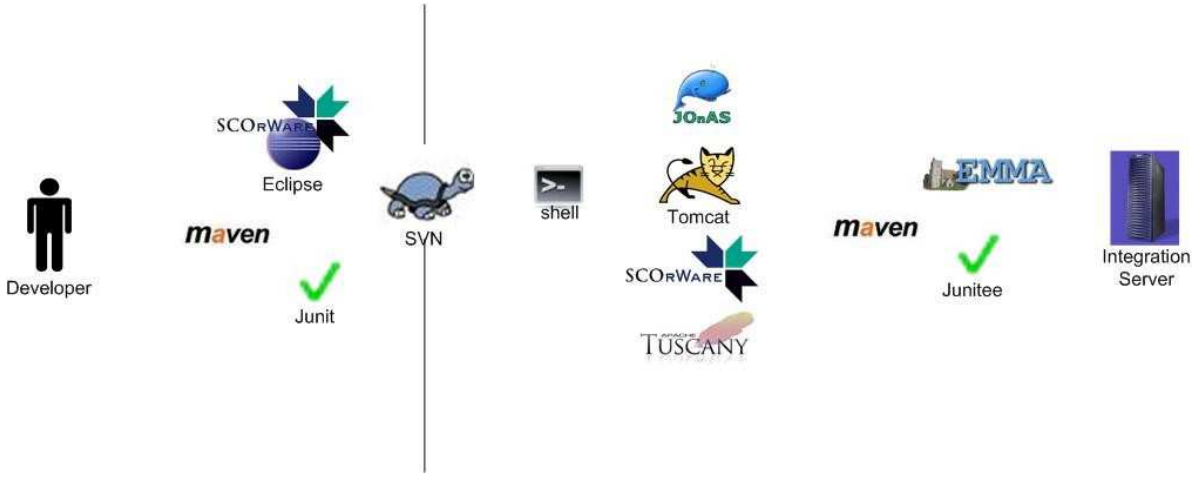


Illustration 43: tools usage – graphical summary

5.4 Technical Specification

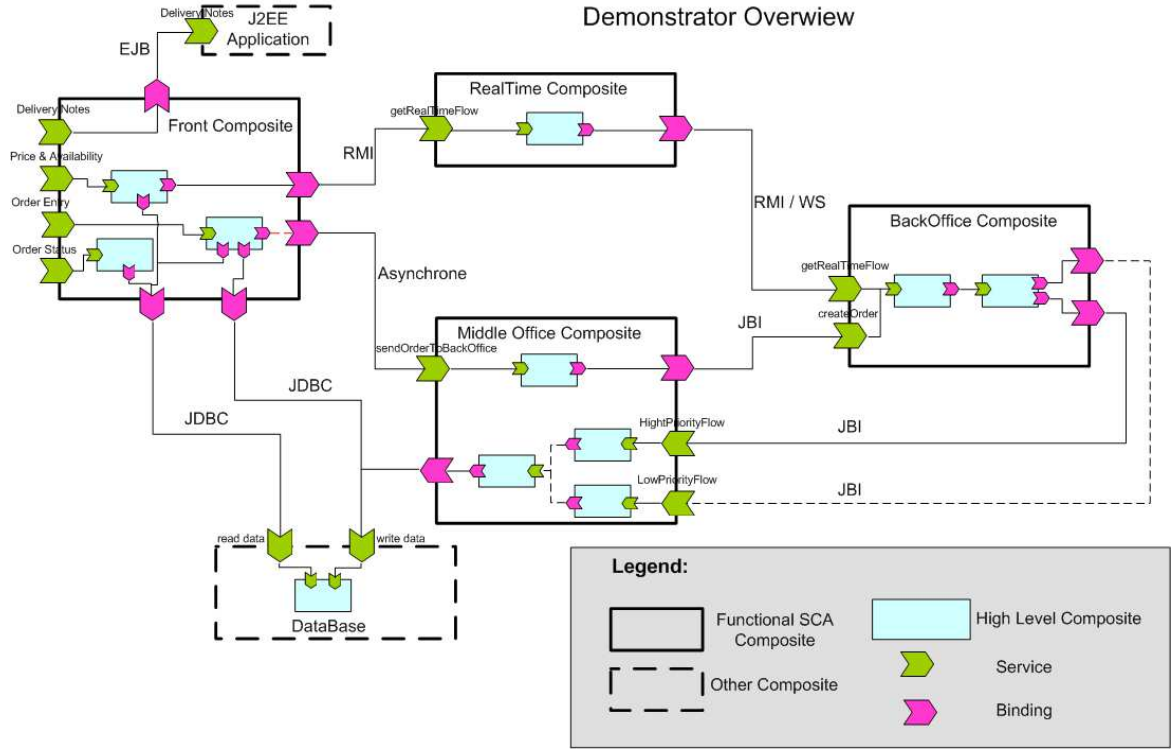


Illustration 44: SCA components of the demonstrator

5.4.1 Technical Components

5.4.1.1 Web Application

The Web Application will be hosted on a Tomcat server. The communication with the Front end services will be done through RMI protocol.

5.4.1.2 Front end services

Depending on the environment, the Front end services will be hosted in one or several SCA containers (Tuscany and/or Tinfi). The communication with Real Time service will be done through RMI protocol. The communication with the Middle Office service will be done through JMS protocol. The communication with Front local database will be done through JDO2 API (using JDBC) or JPA API.

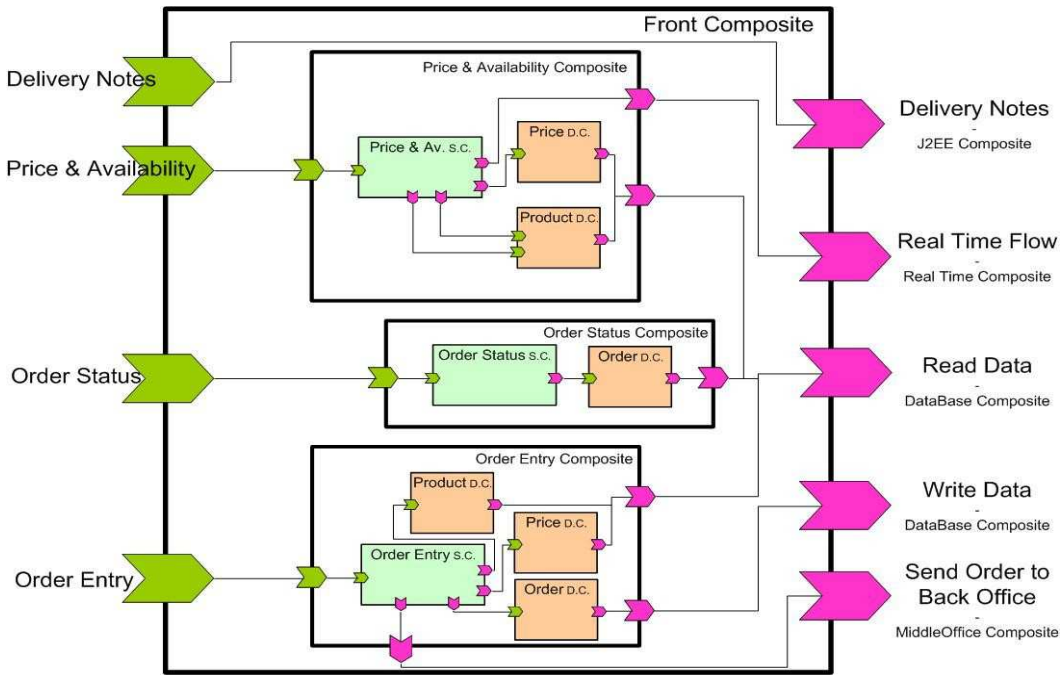


Illustration 45: front composite

5.4.1.3 Real Time service

The Real Time service will be hosted in an SCA container (Tuscany and/or Tinfi).
The communications with Back Offices will be done through Web Service using JBI or with a direct HTTP Client call.

5.4.1.4 Middle Office service

The Middle Office service will be hosted in an SCA container (Tuscany and/or Tinfi).
The communications with Back Offices will be done through Web Service using JBI or with a direct HTTP Client call. The communication with the Front local database will be done through JDO2 API (using JDBC) or JPA API.

5.4.1.5 Front Local Database

Depending on the environment, the Front local database is an Oracle or a PostgreSQL one.

5.4.1.6 Back Offices

The Back Offices are split in two categories: Simulated ones and Real ones. Both have same interface. The Simulated Back Office has no logic and always returns the same result while a Real Back Office (SAP, JD Edwards, CustomERP, ...) is a real Back Office which answers the query done on it. Real Back Offices are out of the Demonstrator scope.

5.4.1.7 JEE Service

The JEE service only contains the Delivery Note business logic. This is a set of EJB 2.1 hosted in JOnAS and accessible through RMI protocol. It manages its own access to the front local database and to the Back Offices (based on our JEE framework).

5.4.2 Technical topics

5.4.2.1 JBI Integration

We think to use a JBI binding in order to integrate

- Middle Office Composite and Back Office Composite

- Real Time Composite and Back Office Composite

This will be allowed by the PEtALS SCA implementation. The idea is to increase the monitoring level of the message going through Middle Office and Back Office.
In fact, in such a case we are more interested in a well monitor message system than an efficient one.

The integration of FraSCAti as a PEtALS service engine has resulted in the definition of a JBI binding (a binding according to the SCA specification). This binding allows one to expose SCA application services as PEtALS services (that is to say as JBI end-points). It also allows an SCA composite to reference any PEtALS service and use it in an SCA application.

Because this binding relies on JBI specifics, it only makes sense in the scope of PEtALS (and potentially of any other JBI ESB).

Notice: in the following lines, when we mention an SCA application / composite running into PEtALS, it means it runs into the SCA service engine based on FraSCAti.

In the context of the demonstrator, using the JBI binding brings additional value on the following points:

- Monitor any remote call made through PEtALS (by using its monitoring console).
- Use PEtALS policies (reliability, security ..) for the communications between different SCA composites running into PEtALS (Middle Office, Back Office Composite and Real Time Composite).
- Enable loosely coupling between the SCA composites running into PEtALS and the referenced services.
    - This way, when a reference changes, e.g. its address or its protocol, we simply need to update the PEtALS configuration. There is no need of modifying or redeploying the SCA composite.

5.4.2.2 Contextual Application

The semantic approach will be used to link references to services.
The more adequate usage should be the links between Real Time, Middle Office and Back Office composites at runtime.
At development time, we will try to illustrate the semantics SCOrWare tools in order to increase the re usability of components by a better knowledge of the already develop components

5.5 Cover of the demonstrator

During the realisation of the demonstrator we will evaluate the SCOrWare tools and the SCA platform for the reuse and enrichment of components. We will compare it with a JEE approach with our actual best practices.
As a result, the demonstrator will be lead by two main axis:

- Reuse and enrichment of components
- SCA and JEE

We will evaluate these axis on each step of a project and framework management:

- Conception
- Development
- Installation
- Production

In this chapter we list all interesting check points we would like to validate.

5.5.1 Conception

Iterative and Retro conception are most in used in EdifiXio project. As a result we will try to point out if SCA and SCOrWare tools leads to any incompatibility of such approaches.
The idea is to see if SCA approach can be easily used with EdifiXio project without any major changes or if such approach imply some radical changes. In this second case, the demonstrator will help us to list all these changes.

Then we will focus on the two following check points for evaluate SCA and SCOrWare tools at conception level :

- Re usability of a concept or a pattern.
- Impact of component scales for development costs and production performances.

5.5.2 Development

Even if the demonstrator will be composed of few SCA components, we will try to illustrate the reality of EdifiXio Framework. This means that we will try to measure the impact on SCA approach in a real industrial context.
As a result, the components management at development time will be point out regarding these criteria:

- Able to deal with more than 200 components
- Cost of adding a components
- Cost of finding an existing component
- Cost for enrichment of an existing component
- Packaging time
- Compilation time
- Learning curve
- Tools dependencies

Then all these data will be compared with our current JEE approach.

The re usability of JEE component into SCA components will be illustrated with the EJB Delivery Note component integrated on the Front Office SCA composite.
The goal of such case is to evaluate the compatibility and relevance of

- using already created JEE components into a new SCA projects.
- Migrating functionality by functionality an existing project into SCA standard.

Then thanks to the Order Entry component, we will stress on a particular technical point: the transaction management and especially two phase commit (see §2.3.4 for more details)

Moreover, we will evaluate the integration of SCA and SCOrWare tools with our existing Tools (see §3.2), API (JDO2, JPA, Struts, Flex, ...) and Framework (Speedo, JOnAS, JORAM, Tomcat, SAP Netweaver, ...). Thanks to that we will evaluate the work to migrate or use the EdifiXio Framework with SCA.

5.5.3 Installation

The cost of an SCA project installation will be evaluated in terms of time and complexity.
The SCOrWare tools impacts will be pointed out.

Even if we won't set up a specific Qualification environment for the demonstrator. We will try to identify any blocking point. For instance, we are expecting that only Environment parameters had to be changed from an Integration environment to a Qualification one (and then to a Production one). The isolation of such Environment parameters from Architecture parameters and Applicative parameters would be an important check point.

As a result the Demonstrator is split in several runtime, running on SCA standalone (the business

service), on Tomcat embedded (presentation layer), on JOnAS JEE (JEE components) and on SAP Netweaver (back office components).

5.5.4 Production

The demonstrator production target simulation should illustrate how SCA and SCOrWare tools may help us to reduce project costs with a better usage of the existing components.
The global performance (time response, resource consumption, ...) will be an important check point.
Another important check point from a production project is the availability to be monitored from a technical point of view (Break down detection, Pool, Memory, I/O, ...) and a functional one (Alert mechanism, data traceability).
Even if we are not expected SCA to have an already implemented monitoring solution, we will expect SCA and SCOrWare tools to easily allow us to plug a monitoring system. For instance, the usage an JBI binding instead of a RMI one should allow us to take benefice of the PEtALS monitoring and traceability message system.

6 ORCHESTATION OF TRANSACTIONS (Task 3.4)

Change history vs. version 1.0: the works done for this usage demonstrator are fully specified in the first version 1.0 (October, 2007) of the specification, as no more work has been done, since the withdraw of the main partner involved in this task, i.e. Amadeus, during the 1st year of the project.

7 NETWORK MONITORING (Task 3.5)

Change history vs. version 1.0: this usage demonstrator has been proposed during the second part of the project, introducing a new industrial partner (Thales Communications), following the withdraw of Amadeus. This part of the final version of the specification presents this new usage demonstrator, not included of course in the first version 1.0 (October, 2007).

7.1 Motivation

Thales is right now moving towards a new business perspective: from product provider, it is moving progressively to product and service provider. This implies an emerging interest in minimizing the cost of integrating different "pieces of software". In these terms, Thales is studying different approaches to bring flexibility and dynamical capabilities into software integration. One of the main topics of interest on which we could apply the SCA technology is network management and monitoring. Typically, these applications cover a set of different but interconnected subjects: Trouble Ticketing, Inventory, Monitoring, Discovery, Fault Management, etc. The main goal of our scenario is to bring some of these modules into the SCA world in order to build components that can be reused, exchanged and easily extended for network monitoring and management purposes. In the scenario a system administrator will manage the network by using the functionalities of four different components. The application will allow the system administrator to:

- Store information about the entities of the network in an inventory. This permits having a list of hosts and services that are currently in the network.
- Discover automatically running hosts and services. Inventory allows to create entities manually, but when deploying new equipment it might be more useful to discover the equipment and then just export the data to the inventory
- Monitor the services running in the hosts of the network. The administrator would have access to a list of alarms generated by the monitoring component. These alarms are generated whenever the status of a service of a host changes.
- Launch these tasks from a management GUI. This interface will show the list of entities of the inventory, allow managing discovery processes and listing the generated alarms.

With this scenario we believe we can prove the effectiveness of SCA applications under the scope of

typical information system management. Acquiring, testing, replacing, upgrading and integrating different software components is a real challenge, which imposes always an important financial cost. They match with SOA and the different approaches proposed by entities like the TeleManagement Forum (TMF). Moreovernot only does the SCA approach match perfectly the SOA approach taken in OSS (Operation Support Systems), but it also fits the crucial requirement of flexible design of various monitoring networks according to changing deployment contexts and their assembly out of the basic standard components that have been outlined above. Finally, SCOrWare-specific enrichments around the SCA standard maximize such flexibility : below the service layer, the integration layer provided by the SCA-compatible PEtALS ESB; above, the business workflow layer provided by the JWT / Scarbo SOA-ready BPM solution; and the methodology and tools developed and tested all along the project.

7.2 Architecture

The architecture can be decomposed in 4 different parts: the discovery module, which is in charge of the discovery of new network elements and services. It will provide info to populate the inventory; the inventory module, which stores information of hosts and services (imported from the discovery or created explicitly); the monitoring module, which notifies detected problems on the hosts and services, and the visualization module, which displays host and service information and the list of generated alerts. There is another element which will play a part in the scenario which is the creation of a virtualized network that will be used to perform both the discovery processes and monitoring.

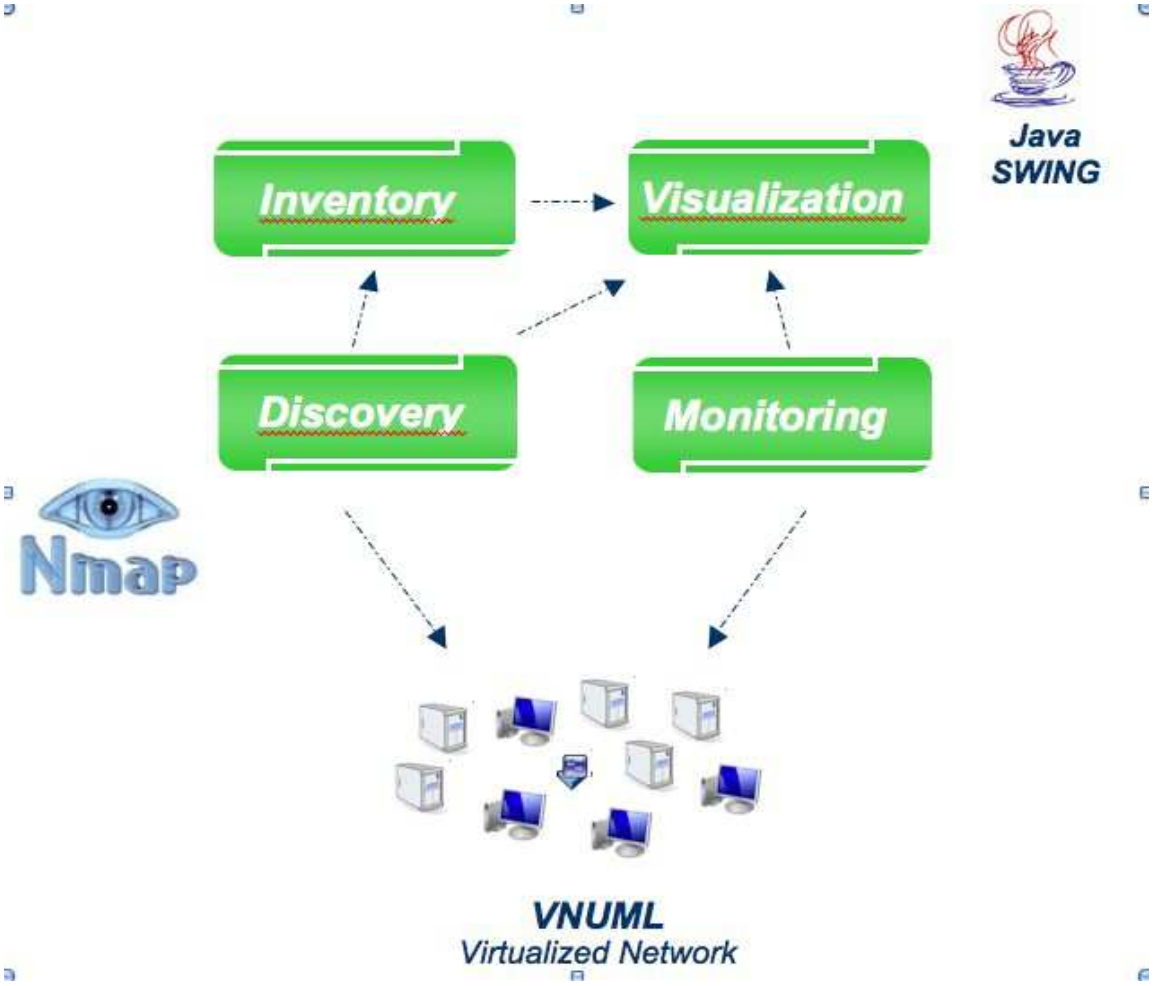


Illustration 46: network monitoring functional architecture

7.2.1 Components overview

As seen in the architecture, the four different modules will be built as four different SCA components.

7.2.1.1 Discovery component

The discovery module will be in charge of performing the discovery of new entities in the network. Entities to discover can be hosts, that is physical computers, or technical services running inside a host, such as http, SMTP, netbios or other services. A discovery agent is in charge of collecting this information and adapting it to the desired format. The export functionality permits obtaining information about discovered network resources in an easy way, which can be used to populate the inventory.

7.2.1.2 Inventory component

The inventory module stores information about network entities found in the information system. This module allows the creation and persistence of entities (also known as managed entities). There are two different ways of creating this entities:

- Using make and create methods: methods for entity creation are generic, so it is possible to create any type of entity supported in the implementation by using the same set of methods.
- Using the import capabilities of the module: by invoking first the export functionality of the discovery module it is possible to import the discovery data automatically into the inventory.

7.2.1.3 Visualization component

The visualization module permits the utilization of all the functionalities available in the other modules. In first place, the GUI shows a graphical representation of the entities in the inventory, hosts and their services. Furthermore it allows creating, updating and removing these entities in an easy way. On the other hand, a console shows information about the events and status of the application. The different menus permit launching discovery process and performing the export/import capabilities.

7.2.1.4 Monitoring component

The monitoring module is in charge of firing alarms whenever a service does not function properly. This information will be sent to the visualization module, so the administrator can react and fix the detected problem. Different types of alert detectors can be set up for monitoring services in different machines.

7.2.2 Workflow overview

There are two typical workflows that can take place in our scenario.

A first scenario would consist in the deployment of a new local area network. The administrator would launch, in first place, a discovery process which will obtain information about all the new equipment found in the network (hosts and their services). Then, by using the export/import functionality of the discovery and inventory modules, the inventory will be populated with information about the discovered entities. Finally, the administrator can show a graphical representation of all the entities thanks to the visualization module. Since the discovery process may not have been accurate enough, the administrator can edit/update information in each entity.

The second scenario would consist in alarm surveillance. At first, the administrator would decide which machines and services should be monitored. Once they are selected, the GUI will alert the administrator of the generation of new alarms. The administrator would then fix the problem manually (no reconfiguration module in the demonstrator).

7.2.3 Architecture evolution

The business of doing network monitoring is a pretty straightforward one. Focusing on the “Deploy new network” scenario, it always involve the steps of getting data about network services first, then

storing it, then manually polishing it. What changes is the context, that is the monitored network(s), that may change over time or be completely new. In order to address this problematic in an efficient manner, we'll demonstrate several approaches to handle (at methodology, design and runtime level) the case of how two Discovery processes deployed in different networks can report to a single inventory.

First this will be done by SCA configuration of a simple additional Discovery service implementation that composes two other referenced Discovery services.

Following this, we'll show how a "deploy new network" workflow process can compose any number of Discovery services.

Then we'll show how such processes can follow changes in the design of SCA services they use, thanks to the bridge built by the STP Intermediate Meta model between the SCA editor that changes them and the JWT Workflow Editor that helps adapt to these changes.

Finally we'll create an abstract template of the "deploy new network" workflow from which any such workflow can be designed for any concrete network(s) to monitor. We'll also study how semantics may help doing the link from abstract to concrete in a business-meaningful way.

7.3 Functional specifications

7.3.1 Discovery component

The discovery module provide two different sets of functionalities: in first place, the ones related to the discovery process management: launching or stopping discovery processes. It is also possible to obtain a list of running discovery processes. Besides, the export functionality permits obtaining the information retrieved during a discovery process.

7.3.2 Inventory component

The inventory module stores information about network entities found in the information system. In these terms, the inventory allows creating, reading, updating and deleting entities. Since these methods are generic, each method allows the creation of different types of entities. Besides, it is possible to retrieve information about the types supported in the inventory.

7.3.3 Visualization component

In first place, the visualization component permits the visualization of network entities in a GUI. Entities are represented in a graph where hosts are interconnected and each technical service is attached to its corresponding host. Moreover, the GUI contains a console which will show information about fired alarms. Finally, the administrator will be able to use all discovery, inventory and monitoring functionalities from the graphical interface.

7.3.4 Monitoring component

The monitoring module will provide information about the alarms generated when monitoring the devices in the virtual network. The component provides lists of alarms, which indicates the current state of a service in a specific machine. Then, thanks to this information, the administrator will verify manually the behavior of a service in order to ensure the performance of the service.

7.4 Technical specifications

7.4.1 SCA Component Diagram

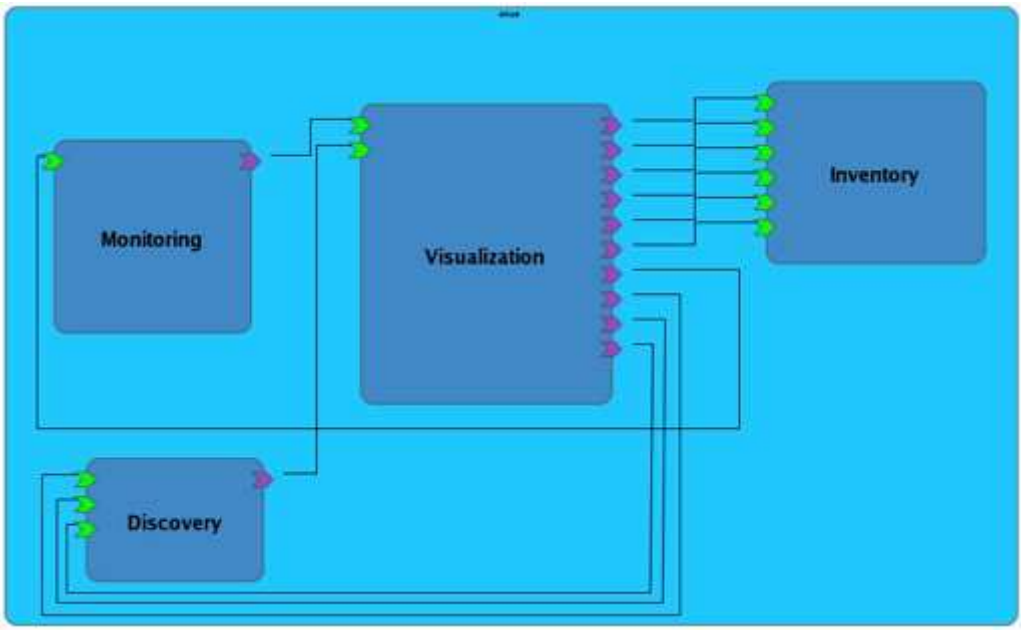


Illustration 47: network monitoring SCA component diagram

7.4.2 Bindings

In our case, we will use a Web Service binding for component communication, since these four modules are usually detached. However, in order to generate the callbacks from the discovery and monitoring components to the visualization, we will use a JMS binding via PetALS.

7.4.3 Provided functionalities

7.4.3.1 Discovery component

This is the list of functionalities provided by the discovery module:

Method	*Parameters*	*Return type*	*Description*
startDiscovery	None	Id: identities the discovery process started	Launches a new discovery process (non-blocking), and returns an identifier of the current process
StopDiscovery	Id: identities the discovery process to stop	Id: identities the discovery process started	Tries to stop the discovery process with the given identifier
exportDiscoveryData	url: location of the generated discovery report	None	Creates a report (XML) file with information of the discovery

7.4.3.2 Inventory component

This is the list of functionalities provided by the inventory module:

Method	*Parameters*	*Return type*	*Description*
Create/read/update/delete Managed Entity	(to be specified per method)	(to be specified per method)	Creates, reads, updates and deletes hosts or services from the inventory
ImportDiscoveryReport	Discovery Report: the discovery report with the information of the discovery	None	Obtains a report (XML) file with information of the discovery
getManagedEntityTypes	None	Array of types	An array with the types supported by the inventory

7.4.3.3 Visualization component

The visualization exposes as services all the services from other components (so references are reachable from the application) that have to invoked by the GUI. Apart from these services, the visualization component needs two callbacks in order to update the graphical interface responding two different events:

Method	*Parameters*	*Return type*	*Description*
OnDiscoveryProcessFinished (reference from the Discovery Module)	A text representation informing of the end of the discovery process	None	The method is invoked when a discovery process is finished. A notification message is shown in the GUI
OnNewAlarm (reference from the Monitoring Module)	The created alarm	None	The method is invoked when a new alarm is created. The alarm is shown in the alarm list

7.4.3.4 Monitoring component

This is the list of services provided by the monitoring module:

Method	*Parameters*	*Return type*	*Description*
getAllAlarms	None	A list of Alarms	Returns a list of all generated alarms
getAlarmsByTemplate	None	A list of Alarms	Returns a list of all alarms generated that match the given alarm

7.4.4 Workflows

7.4.4.1 "Deploy new network" workflow and wizard

Open Wide adds a new "Tool" sub menu action "Deploy new network" to the OSSj GUI that shows

the GUI to input necessary network discovery information and then executes a configured "Deploy new network" workflow. Open Wide designs such a workflow using JWT WE by letting it call the default discovery service and then the default inventory service using Web Services.

7.4.4.2 "Handle alarm" workflows

Open Wide will define a workflow designed to handle monitoring alarms. This workflow alerts an administrator, then allows him to manually fix the problem in a manual workflow task, then notify listeners. To implement this use case, the administration interface of the Nova Bonita workflow engine will be used as UI.

Open Wide will study real time notification to start and end the workflow, using the PEtALS JMS binding integrated in the demonstrator by EBM WebSourcing.

7.4.5 Evolution of the architecture

Here is how Open Wide implements said different approaches of how two Discovery processes (and more) deployed in different networks can report to a single inventory.

7.4.5.1 SCA-composed Discovery services

Simple SCA configuration will achieve it with the help of an additional DiscoveryService Java implementation that calls successively two other SCA-injected DiscoveryService references. This will be done by Thales with the help of Open Wide.

7.4.5.2 Workflow-composed Discovery services

Then several "deploy new network" workflow process will be designed using JWT to compare designing calls to SCA-defined services versus Web (WSDL defined) services.

7.4.5.3 Following service design changes in JWT processes design

To achieve this, Open Wide develops in JWT synchronization of a JWT model's service applications with services managed by a given underlaying STP IM model. The underlaying STP IM model being used is configured in a JWT EMF Aspect meta model extension of the JWT model. Synchronization is started by a JWT external Action. Existing service applications are recognized by their configuration (type, service interface, ...), missing ones are created, deleted ones are traced through warnings.

Once done, Open Wide showcases it in the following use case : the composite configuring the default DiscoveryService implementation being edited in the SCA editor, its service definition is changed, then (through the SCA Editor's integration with STP IM) this change is published in the common underlaying STP IM model ; then the previous "Deploy new workflow" being edited in JWT, the user synchronizes JWT service applications with services managed by the common underlaying STP IM model, and changes the workflow as necessary.

7.4.5.4 Abstract semantical workflow template

Finally we'll create an abstract template of the "deploy new network" workflow from which any such workflow can be designed for any concrete network(s) to monitor. We'll also study how semantics may help doing the link from abstract to concrete in a business-meaningful way.

7.5 Coverage

7.5.1 FraSCAti and SCA Editor

Java implementation, interface and annotations, WS and SCA binding, Service, Reference in all components ; included composite in "SCA-composed Discovery services" use case

7.5.2 FraSCAti-integrated PEtALS

The SCA Service Engine enables integration of FraSCAti in PEtALS. Features of this component are:

- Instantiate SCA components using FraSCAti during deployment.
- Expose the services of the SCA composite to the NMR
  - Unmarshall incoming message exchanges to Java method call.Marshall returning results in message exchanges.
- Call distant references of the SCA composite as JBI endpoints.

Such an integration implements delegation of PEtALS the management of binding references in SCA composites. Such a delegation conforms to the "spirit" of SCA as it decouples component implementations from bindings: component implementations are run by the SCA runtime, the SCA Service Engine, whereas binding are bound by PEtALS binding components.

In the context of the demonstrator, using the SCA service engine brings additional value on the following points:

- Generate the callbacks from the discovery and monitoring components to the visualization, using the JMS binding component.
  - The discovery and monitoring composites send notification at a JMS queue using the SCA reference with JMS binding.
  - The visualization composite has a service with a JMS binding to receive notification from a JMS queue.
- Monitor any remote call made through PEtALS (by using its monitoring console).
- We can use PEtALS policies (reliability, security ..) for the communications between different SCA composites running into PEtALS (discovery, monitoring and visualization composite).

7.5.3 JWT and Scarbo

- Workflow business and technical design, meta model extensions (aspects), integration with SCA editor, calls of WS and SCA services, execution as packaged XPDL on Scarbo (TaskEngineFramework, FraSCAti, Nova Bonita) runtime in all workflows.
- Synchronization with SCA Editor through STP IM in "service design evolution" use case, ontology view and semantic resolution of abstract annotated SCA services in "abstract workflow template" use case .

7.6 Methodology and results

7.6.1 Project layout and SOA management

SOA management revolves around managing service definitions and their uses (implementors and clients). In this demonstrator, decoupled remoting is achieved mainly through Web services (which FraSCAti supports through its CXF integration). Therefore being SOA implies that we have to manage their WSDL interfaces. Moreover, one of the benefits of FraSCAti as a native Java SCA

implementation is that it allows developments to be done fully in the Java world . Therefore to take advantage of it, we'll design services in a Java-first way and manage both their Java interface and generated WSDL.

The layout of (maven-built and assembled) development projects mirrors such SOA requirements.

Domain services (i.e. "the Monitoring SOA") are all defined in the monitoring-API project. They are defined as Java interfaces, WSDL is generated from there, but these Java interfaces can be used in SCA composites as such ones.

Each service implementation implies at least two projects. Let's takethe Discovery process implementation as an example:

- monitoring-discovery-impl contains application-specific local implementation code of the service, as well as its unit tests (using annotated Junit4)
- monitoring-discovery-sca contains SCA files configuring the actual deployment. Remote exposition of the service uses mainly WS bindings, and required remote services are injected in them thanks to WS bindings as well. Here are also integration, and finally the Launcher of the implementation (does a FraSCAti standalone startup).

Other projects can provide JBI artifacts (Service Units, Service Assemblies) required to deploy PEtALS specific features of the demonstrator. Finally maven_repo is a private maven repository to put any kind of non-public maven dependencies required by implementations.